NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE IMPACT OF VERBAL REPORT PROTOCOL ANALYSIS ON A MODEL OF HUMAN-COMPUTER INTERFACE COGNITIVE PROCESSING

by

Barbara L. Treharne

March 1991

Thesis Advisor: Kishore Sengupta

Approved for public release; distribution is unlimited.

T254121

Unclassified
ECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

a REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b RESTRICTIVE MARKINGS

a SECURITY CLASSIFICATION AUTHORITY
Multiple Sources

3 DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release; distribution is unlimited.

b DECLASSIFICATION / DOWNGRADING SCHEDULE
(OADR)

PERFORMING ORGANIZATION REPORT NUMBER(S)

5 MONITORING ORGANIZATION REPORT NUMBER(S)

a NAME OF PERFORMING ORGANIZATION
Naval Postgraduate School

6b OFFICE SYMBOL (If applicable)
367

7a NAME OF MONITORING ORGANIZATION
Naval Postgraduate School

c. ADDRESS (City, State, and ZIP Code)
Monterey, CA 93943-5000

7b ADDRESS (City, State, and ZIP Code)
Monterey, CA 93943-5000

a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b OFFICE SYMBOL (If applicable)

9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

c. ADDRESS (City, State, and ZIP Code)

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

1 TITLE (Include Security Classification)
THE IMPACT OF VERBAL REPORT PROTOCOL ANALYSIS ON A MODEL OF HUMAN-COMPUTER INTERFACE COGNITIVE PROCESSING

2 PERSONAL AUTHOR(S)
Treharne, Barbara L.

3a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	March 1991	104

6 SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

7 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Human-Computer Interaction, User Interface, Cognitive Complexity, Verbal Protocols

9 ABSTRACT (Continue on reverse if necessary and identify by block number)

This exploratory study used the "think-aloud" protocol to demonstrate the effectiveness of Kieras and Polson's Goals, Operators, Methods and Selection Rules and the Cognitive Complexity Model. An experiment comparing the cognitive processes of users on two file management interfaces, a Command Language and Direct Manipulation interface, was conducted. The think-aloud process was chosen as the methodology for conducting this experiment because of its insights into the user's perceptions of both the task and device representations. The experimental results provide implications for the study of cognitive processes--the nature of the interface design influences the users' mental models of a system, which has a direct affect on the users' performance on a given interface. This methodology also provides an evaluation technique which may improve the design process of the user interfaces. Finally, the results support the think-aloud protocol as an effective evaluation tool of user interface designs.

0 DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION
Unclassified

2a NAME OF RESPONSIBLE INDIVIDUAL
Kishore Sengupta

22b TELEPHONE (Include Area Code)
(408) 646-3212

22c OFFICE SYMBOL
AS-SE

D Form 1473, JUN 86 *Previous editions are obsolete*

S/N 0102-LF-014-6603

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Approved for public release; distribution is unlimited.

The Impact of Verbal Report Protocol Analysis on a Model of Human-Computer Interface Cognitive Processing

by

Barbara Lynn Treharne
Captain, United States Army
B.S., United States Military Academy, 1980

Submitted in partial fulfillment of the requirements for the degree of

MASTERS OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1991

ABSTRACT

This exploratory study used the "think-aloud" protocol to demonstrate the effectiveness of Kieras and Polson's Goals, Operators, Methods and Selection Rules and the Cognitive Complexity Model. An experiment comparing the cognitive processes of users on two file management interfaces, a Command Language and Direct Manipulation interface, was conducted. The think-aloud process was chosen as the methodology for conducting this experiment because of its insights into the users' perceptions of both the task and device representations. The experimental results provide implications for the study of cognitive processes--the nature of the interface design influences the users' mental models of a system, which has a direct affect on the user's performance on a given interface. This methodology also provides an evaluation technique which may improve the design process of the user interfaces. Finally, the results support the think-aloud protocol as an effective evaluation tool of user interface designs.

1923
C.1

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

Despite the growth in computer technology in today's society, the potential for computer productivity and effectiveness has not been reached. A primary reason for this is that human-computer interfaces are not designed to be compatible with the way in which people think--many still experience fear and anxiety when required to interact with a computer, resulting in their inability to learn the system and unnecessary errors during usage [Ref. 1:pp 351]. The purpose of this thesis is to provide new insights into cognitive processes of users when interacting with computers. Ultimately, this knowledge can be applied by designers to make systems more usable by altering the behavior of the device, and thus matching its behavior to the user's cognitive process of task accomplishment [Ref. 2:pp 365]. The designer can increase usability in several ways, such as reducing the complexity of task structures [Ref. 2:pp 366] or applying the acquired knowledge to the generation of training material or reference documentation.

Currently, there exists a number of models which theorize, from different aspects, the cognitive processes which a user undergoes while performing certain types of tasks through

interaction with the computer system. Two of the more prominent models are the Goals, Operators, Methods and Selection Rules and the Cognitivity Complexity Model.

These model theories have been developed based upon various methods of experimentation--such as, analysis of keystrokes, user logs, error analysis, retrospective verbal reports, questionnaires, and, rarely, using the concurrent verbal reporting "think aloud" methodology. All of these sources of data are required to complete the "big picture" of user-interface cognitive processing. Therefore, the results of each methodology must be fully integrated with existing theories of cognitive models. Currently, there is insufficient research in the area of verbal protocol analysis. In order to gain consistency, validity and full integration of experimentation results, more studies/experimentation conducted using verbalization protocols must be completed. This thesis will assist in correcting this gap in research by providing an experiment using verbal protocol analysis in order to model users' cognitive processes.

B. USER INTERFACES

While there are a number of user interface styles available today, the two most common are the Command Language interfaces and the Direct Manipulation interfaces. These interfaces are described in general below.

Command Language Interfaces require the user to communicate with the computer by typing a formal language, using a specific syntax. The user in required to learn and memorize the commands and the sequences needed to complete an operation within a given task [Ref. 3:pp 154].

Direct Manipulation Interfaces enable the user to communicate with the computer, and thus control activities, through direct action on visible objects rather than by the use of a procedural language. The user is provided a continuous representation of the objects and actions of interest, while allowing the user to execute the command through actions such as movement and selection (performed by pointing and clicking) of objects with the mouse of joystick [Ref. 3:p. 154].

To date, numerous studies and experiments have been conducted in attempts at comparing the two interfaces. Margono and Shneiderman conducted an experiment which compared the file manipulation operations on the Apple Macintosh, a Direct Manipulation interface, and the IBM PC with MS-DOS, which uses the Command Language interface [Ref 3:pp 154]. This 1987 experiment measured the mean times and error rates of the subjects. It concluded that the Apple Macintosh was more user friendly--because it is easier to learn and use due to only modest memorization requirements and because it takes less time to perform tasks on the Macintosh due to the

elimination of typing with the mouse and pull-down menus. Karat, Fowler, and Gravelle conducted a study to examine learning and performance differences between a Command Language and a Direct Manipulation interface [Ref. 4:pp 489]. This study measured the performances of novice users tasked to perform routine file management operations on each of the two interfaces. It substantiated the ease of use and learning of the prototype Direct Manipulation language. However, it was not able to make accurate predictions from the production system models generated to lend support to the GOMS model. This failure was seen as a failure to encompass error recovery behavior into the GOMS approach, not as a complete condemnation of the GOMS framework. In Te'eni (1990), experimental subjects were broken down into three groups--one with no feedback, one with traditional dialogue feedback, and one with visual Direct Manipulation feedback--and given two complex tasks of providing input to the final course grade. The groups were measured by mean performance times and logical error rates. This study demonstrated that feedback resulting from Direct Manipulation is more effective and time efficient than the distance form of feedback in conditions of high complexity [Ref. 5:p. 1-25].

C. GOAL OF STUDY

This thesis involved conducting an experiment in which users performed a set of complex tasks while interacting with the two separate computer interfaces--the Direct Manipulation and Command Language interfaces. The experiment involved recording the users as they "think aloud" throughout the performance of routine tasks, which required direct interaction with the human-computer interface of a file management system. The users verbalized their thought processes concurrently with the performance of the task. By doing so, the user described information required to perform the task. It not only traced the actions of the users, but indicated why each action was taken. This "why" information fills a void left by other experimentation methodologies by providing a direct insight into the user's true perceptions of the task and device representation [Ref. 6:pp 3-4]. These verbalizations and the corresponding user logs of the user interaction with the computer are then transcribed and analyzed with respect to the premises of the GOMS and Cognitive Complexity models.

II. THEORETICAL PREMISE

A. USER COMPLEXITY

The complexity of user interfaces involves the complexity of the entire interface system from the user's perspective [Ref. 2:pp 365]. It pertains to the complexity of the device itself, the tasks being performed, and the difficulty of learning to operate the device. In order to significantly reduce the complexity of any of the above components of a user interface, a designer must have a basis for an analysis and determination of the relative complexity of the user's operating environment. The purpose of this research is to further the knowledge of interface designers by providing evidence that continues to support the use of the GOMS and Cognitive Complexity models as theories to measure and reduce the complexity of user interfaces.

B. COGNITIVE COMPLEXITY AND THE GOMS MODEL

The Cognitive Complexity Theory and GOMS model incorporate the user's representation of the task to be performed, the user's representation of the device used to operate and perform the task, and the relationship between the task and the device in order to quantify the amount and complexity of knowledge required in operating a system. Each

of these components is vital to the full understanding and development of human-computer interface activities that are easy to use, efficient, and error-free.

The GOMS Model prefaces its framework on the user's mental information processing capabilities by describing a Model Human Information Processor. This processor consists of three interacting subsystems which work together to delineate the user's mental procedures for performing a single action. These three subsystems are: the perceptual system, the cognitive system, and the motor system. The perceptual system carries sensations of the physical world detected by the body's sensory systems into internal representations of the mind [Ref. 7:pp 25]. The cognitive system then connects inputs from the perceptual system to the right outputs of the motor system [Ref. 7:pp 35]. Within the perceptual system, there are two important types of memory--working and long-term memory. Long-term memory consists of a body of knowledge, gained through repeated associations with that knowledge, that is stored in a network of related chunks of memory. Working memory is the activated portion of long-term memory which responds to the input detected by the perceptual system. An important aspect of the cognitive system is the recognize-act cycle which similates the fetch-execute cycle of familiar computer operations. Once recognized, the contents of working memory initiate actions associatively linked with them in

long-term memory; these actions, in turn, modify the contents of the working memory, ultimately resulting in the requirement of an output [Ref. 7:pp 27-39]. Thought is transferred to actions--actions such as performed by the head-eye system and the arm-hand-finger system--through this iterative process of recognize-act cycles. These actions are controlled by the motor system [Ref. 7: pp 34]. Within this specific architecture, each component system is assigned parameters, such as cycle time or storage capacity, used in the derivation of predictive models for cognitive complexity measurements.

To further amplify this model and to elicit more direct correlation between the user's mental model and the relative complexity of the user's cognitive process, the family of models known as the GOMS model was developed. This model is based upon the "rationality principle" which states that a person attempts to achieve goals by doing those things that the task itself requires to be done [Ref. 7:pp 86]. The model further assumes that behavior can be described as a sequence of a small number of information-processing "operators" which can be both described and timed. Based upon the validity of the rationality principle and the existence of such operators, it is feasible then to predict the sequence in which the operators will be performed [Ref. 7:pp 139]. These operations and their sequences are performed in alternating iterations of the recognize-act cycles of the cognitive system within the

Model Human Information Processor. By knowing this sequence, and thereby knowing the number of operations and production rules required to perform a given goal, or task, it will consequently determine the relative complexity of that goal.

The GOMS model consists of the following components: a set of goals, operators, methods for achieving those goals, and selection rules for choosing the method to be used. The premise of the model is that goals are performed in a hierarchial manner in accordance with the rationality principle. The goal structure can be considered the plan for carrying out the completed task. The operations are mental representations of elementary functions that the device can perform. Methods are learned procedures that the user already has at performance time, i.e., a skill, and that are needed to satisfy a specific goal or subgoal--it can be as simple as a single keystroke, or as complex as a complete set of subgoals and operations necessary to meet its higher level goal. The selection rules specify which methods should be used to satisfy a given goal and can be correlated to a condition-action (if-then) pair [Ref. 2:pp 366].

The GOMS model relates directly to the user's task representation--it is through knowledge of the task that the user must establish goals, methods, selection rules, and operations necessary for the performance of any given task. The job-task environment can be viewed through a production

system which is based upon the theory that mental processes, or behavior, can be represented as a set of specific response actions, each made in response to a particular stimulus condition--i.e., a condition-action pair as seen as a selection rule within the GOMS model [Ref. 2:pp 369]. The production system is composed of a collection of production rules (selection rules) and a working memory. Within working memory, there exists representations of the system's current goals, information about the status of current and past actions, and representations of inputs from the environment [Ref. 2:pp 369]. The production systems operate in alternating states of the recognize-act mode. During the recognize mode, the conditions of all the productions are tested against the contents of working memory. When a condition is satisfied, the system goes into the act mode to execute the operations. The contents of working memory are modified at this stage and the system then returns to the recognize state where more conditions may be satisfied upon the updated working memory [Ref. 2:pp 369]. The theory states that the complexity of the task corresponds directly with the number of production rules that must be acted upon to perform a given task. Therefore, by counting the number of production rules executed, the complexity of any given task may be quantified.

Inherent in defining the complexity of the entire job-task environment is the user's model of the device which must be considered along with the job-task representation. A user's device model is the person's understanding of the internal structure and functions of a device--this is often referred to as "how-it-works" knowledge and has effects on a user's ability of learn and operate a device [Ref. 2:pp 377]. The device representation must characterize the interaction between the user and the device and, subsequently, an explicit and formal representation of the device itself is needed. To do so, this model utilizes the general transition network (GTN), consisting of nodes, which represent states, interconnected by arcs, which represent possible transitions between states. An arc may consist of a condition-action pair and a specified next state. Nesting, which allows one GTN to call another in a manner similar to subroutine calls in ordinary programming languages, can appear in three places: in conditions, in actions, and in states [Ref. 2:pp 381-383]. Each function performed by a device must be adequately portrayed in a GTN, demonstrating the necessary operations to be formed and the hierarchy of transitions to be executed in sequence.

Once the device representation GTN and the GOMS model/ production system goal structure are defined, the relationship between the characteristics of the user's task and the device

being used in the task may be established. The hypothesis is that a good task-to-device mapping is one in which the goal structure and the device structure graph correspond--thus facilitating ease of learning and operations [Ref. 2:pp 387]. An analysis of a mapping in which there did not exist a correspondence between the GTN and the goal structure graph can be used to improve the overall interface--by either altering the device itself to correspond with the user's goal structure or possibly altering the documentation of "how-it-works" knowledge can be provided to alter the user's goal structure [Ref. 2:pp 390].

This model provides a formal architecture for analysis of human-computer interfaces. The model addresses the user's mental model, the user's representation of the task, the device model representation, and the relationship between the task and the device on which the task will be performed.

There are numerous studies which have been conducted which offer support for the Cognitive Complexity and GOMS model. Three studies demonstrating the most direct correlation between the theory and the experimental results are addressed here. In 1984, Kieras and Bovair used this theory to analyze the results of a transfer of learning study in which users learned to operate a control device through written instructions. The experiment demonstrated that the production rule representation accurately predicted the

relative difficulty of a set of related procedures [Ref. 8:pp 507-524]. Similarly, Polson and Kieras successfully predicted learning, transfer and execution times of text editing tasks based upon the number of recognize-act cycles required to perform tasks in their 1985 experiment, "A Quantitative Model of the Learning and Performance of Text Editing Knowledge." [Ref. 9:pp 207-212] Finally in 1990, Bovair, Kieras, and Polson used production rule models to make quantitative predictions for both ease of learning and ease of use. A production rule model was developed for a simulated text editor. The model was then evaluated--by comparing its' prediction of learning and execution times with the actual experimental data. The production rule model successfully predicted both learning and execution times, thus providing strong support for the Cognitive Complexity Theory [Ref. 10:pp 1-48].

C. THINK-ALOUD PROTOCOL

In order to facilitate a complete understanding of the implications of the user-interface decision, a complete picture of the human-computer interaction must be acquired. This can only be accomplished if all of the design factors--experience of designers, current trends of input/output technology, ergonomics (human-factors) research, cognitive psychology, and evaluation of working systems--are considered

and integrated into the formulation of well-designed computer interfaces. Reducing the cognitive complexity of user interfaces is essential to achieving the final goal [Ref. 11:pp 1].

The think aloud protocol is a method for studying mental processes which reflects the user's perception of task and device representations. It provides concurrent, spoken comments as participants work through a task. This type protocol is a record of the natural use of software and its aim is to get users to identify problems by explaining what they are trying to do, and why, as well as what problems they are having while doing it. The verbalized protocol is later transcribed and analyzed to provide specific feedback of critical instances of behavior [Ref. 11:pp 18].

Much of current research relies on evaluating interfaces and determining cognitive complexity by evaluating the execution times of task performance, as well as by identifying number and type errors committed. The think aloud protocol supplements this other research of cognitive complexity by providing the reason why specific problem areas/errors are occurring. Key advantages of using the think-aloud methodology is that it quickly pinpoints problems that might otherwise go undetected [Ref. 6:pp 3]. In support of this theory, Wright and Monk (1991) conducted two studies using the think-aloud protocol. Their initial study included an experiment in

which the success of trainee software engineers was compared with that of more experienced designers in evaluating a menu-based interface. They followed this study by one in which designers of a system were compared with designers unfamiliar to that system in predicting and evaluating problems with the system. Monk and Wright concluded that user testing with think-aloud methods not only is an effective technique for interface designers, but also significant gains were to be had from designers carrying out their own evaluations [Ref. 13:pp 255-257].

More importantly, this verbal protocol provides insight to the user's perceptions of both the computer system that he is working on and of the task required of him. By verbalizing his thoughts, the user's task representation (i.e., goal hierarchies) and the user's device representation will be made known. Through this, the task-to-device mapping can be more accurately achieved through the detailed observations provided by the think-aloud protocol--thereby, pinpointing aberrations in the task-to-device mapping, which can be later analyzed to develop recommended alternatives that will provide better task-to-device mapping--thus reducing the overall cognitive complexity of the tasks to be performed on the selected interface design.

D. COGNITIVE COMPLEXITY AND USER INTERFACES

Ultimately, the chosen interface design will impact the user's ability to perform required tasks on the interface. The extent to which the interface allows the user to interact unhampered is a reflection of the demands of cognitive processing imposed on the user. The GOMS and Cognitive Complexity models theorize a method for identifying and reducing these demands. Each of the two interfaces chosen for this research, Direct Manipulation and Command Language, appear to have complimentary demands on cognitive complexity issues with regard to interface design. It is the premise of this research that the Direct Manipulation interface is easier to learn and operate because it has less taxing demands on the user's cognitive processing behaviors. A comparison of the types of cognitive processes required by each of the two interfaces follows.

Because the Command Language interface requires the user to communicate with the computer by typing a formal language, using a specific syntax, the user not only has to learn the requirements of the task domain to be performed, he must also have an understanding of the computer itself. The user is required to learn and memorize the commands and the sequences needed to complete an operation within a given task before he can begin typing them into the computer. The user may make errors due to: the confusion of using the syntax itself,

typing errors, and a mismatch between the user's intention in the task domain with the computer concepts or syntax [Ref. 3:pp 154-155].

Specifically, this approach to interface design has several disadvantages: the commands are difficult to memorize and are therefore error prone; uncertainty exists that a command has been executed as expected (requires the user to follow-up by performing a subsequent command); and the inability to scroll through directories in both the forward and backward direction necessitates a high degree of memorization and/or the repeating of the operation [Ref. 3:p 155]. The heavy load of memorization required by the Command Language interface burdens the Model Human Information Processor with a large number of retrievals of information from long term memory and repetitious recognize-act cycles. As theorized by Kieras and Polson, this correlates directly with a bad task-to-device mapping, and, consequently, higher user complexity.

Direct Manipulation interfaces enable the user to communicate with the computer, and thus control activities, through direct action on visible objects rather than by the use of a procedural language. The graphic interface and mouse selection provides the user with Direct Manipulation interaction and allows the users to operate intuitively--without a lot of memorization. In Direct Manipulation, the visual

representation should match the way in which people think about the problem [Ref. 3:pp 154]. This further allows the user to focus on the task itself with little need to "learn" the computer operating system which is given [Ref. 3:pp 155].

Another central issue in determining the demands placed on the cognitive processors of the user is the feeling that the user has control of the actions he is performing on the computer--this is termed "directness." It is theorized that if the relationship between the command and the action can be made more immediate and direct, the user's understanding will increase [Ref. 3:pp 154]. This relates directly with the recognize-act principles of the GOMS and Cognitive Complexity model in that if the user can more immediately recognize the state that the computer and/or task is in at a given time, the user will be able to select the proper methodology and production rules for the next step within the goal hierarchy. If this is done with more accuracy, there will be less required recognize-act cycles and less production rules fired. The Command Language interfaces create a feeling of indirectness between the user and the world of action. This occurs because the user is constantly describing, through typed commands, the actions rather than actually performing the actions. On the other hand, in Direct Manipulation interfaces, the user performs actions on the objects of interest and the system shows the actions that are performed immediately

[Ref. 3:pp 154]. Therefore, the user has a feeling of control over the objects in the task domain--i.e., the user senses the "directness" of his interaction with the computer. This sense of directness should ultimately result in increased efficiency for the Direct Manipulation interface user.

III. RESEARCH METHODOLOGY

A. INTERFACE DESCRIPTION

The two interfaces, Command Language and Direct Manipulation, have been developed for the purposes of experimentation and are only functional with respect to the operating system's file management system. The system described herein simulates the common functionalities of any typical file management system.

The file management system supports the user in managing the files that work together to produce the program that the user sees and interacts with on the screen. A directory contains the files which make up the program. Subdirectories are utilized to further segregate files in logical groupings for easier identification. A typical file management system allows the user to perform the following functions: to create, copy or move files and to create or delete directories. Each of the two experimental interfaces provide these specific capabilities: access to a directory tree and help screen, create a file, copy a file, rename a file, sort files, delete a file, create a directory, and delete a directory. In addition, the Command Language interface has a command to view the contents (files and subdirectories) of a designated directory. This function is provided automatically by the

Direct Manipulation interface when a directory or subdirectory is selected with the mouse.

At Table 1, is a complete listing of all the commands available within either interface, to include the sequence of steps required to perform each.

Instruction sets given in Appendices A and B were provided to the user in the conduct of the exercise and also provide a detailed description of each of the interfaces as described below.

1. **Command Language Interface**

The Command Language Interface includes the three windows illustrated at Figure 1: (1) a Directory Window which displays the directory tree, (2) a File Window which displays the subdirectories and files stored in a specified directory as identified in the label, and (3) a Command Window which displays user-input commands, as typed on the keyboard.

Figure 1. Command Language Window Format

TABLE 1 - PROCEDURES REQUIRED TO EXECUTE COMMANDS		
Command Name	Command Language Procedures	Direct Manipulation Procedures
1. Copy File	COPYFILE <directory\from filename> to <directory\to filename>	- Select directory - Select filename - Select new name - Select Copy File icon - Accept path name
2. Create File	CREATEFILE <directory\new filename>	- Select directory - Select new name - Select Create File icon - Accept path name
3. Remove File	REMOVEFILE <directory\old file name> to <directory\new file name>	- Select directory - Select file name - Select Delete File icon - Accept path name
4. Rename File	RENAMEFILE <directory\old filename> to <directory\new filename>	- Select directory - Select file - Select new name - Select Rename File icon - Accept path name
5. Sort File	SORTFILE <directory> by <sort mechanism>	- Select directory - Pull down menu from Files Sort window - Select sort mechanism
6. Create Directory	CREATEDIR <directory\new filename>	- Select higher order directory - Select new name - Select Create Directory icon - Accept path name
7. Directory Tree	DIRTREE <directory>	- Select Directory Tree icon
8. Remove D: :tory	- Remove directory contents - REMOVEDIR <directory>	- Remove directory contents - Select directory - Select Delete Directory icon - Accept path verification
9. List Files	LISTFILES <directory>	- Select directory
10. Help	HELP <command name>	- Select Help icon - Select command name

If a portion of any window or screen is obstructed from the viewing screen, the user is prompted to type "M" for more.

The keyboard is the only input device used for the Command Language Interface. All commands are typed on the keyboard at the command line. Only one syntactical format is correct for each command. If the user types a command incorrectly, omits an argument, or specifies a non-existing file, an error message appears in reverse video at the base of the command window as shown in Figure 2.

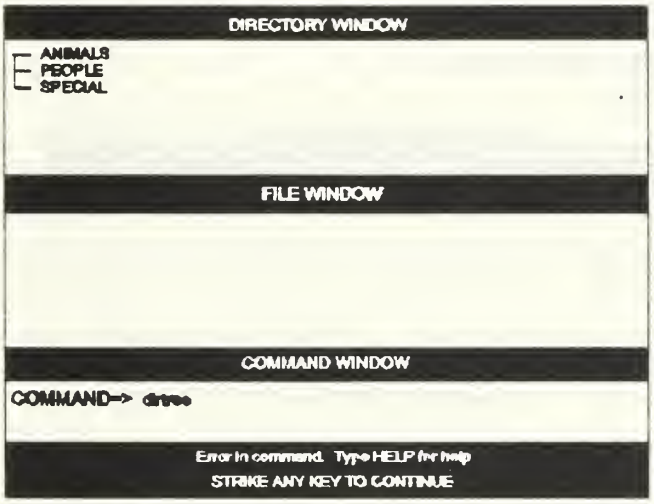


Figure 2. Command Language Screen Containing Error Message

The error message indicates the type of error made, but not the corrective action to be taken.

A user can access a general Help screen at any time by typing the command HELP at the command line of the Command Window. The user can obtain a detailed description of each command by typing the command HELP, followed immediately by the command name. Figure 3 shows an example of the Help screen for the CREATEFILE command.

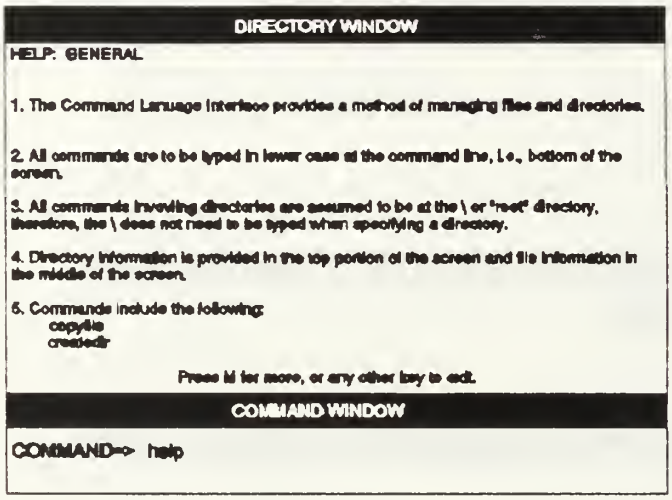


Figure 3. Command Language General Help Screen

An example of the Command Language screen is provided at Figure 4 as the user might see it during the performance of a complex task. In this instant, the user has displayed the directory tree and listed the files in the 'business' directory.

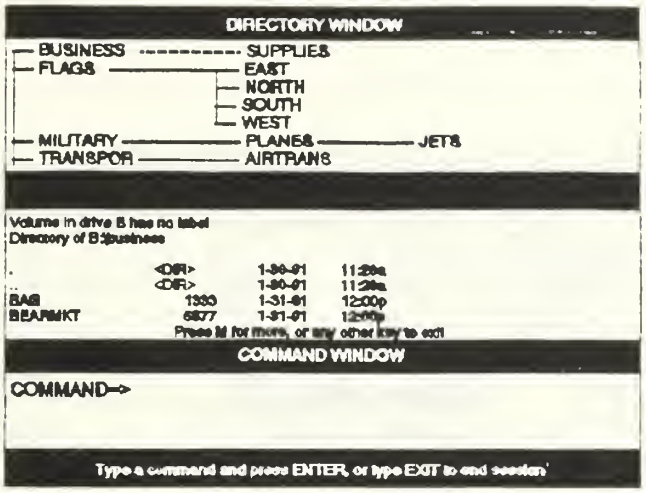


Figure 4. Command Language Screen
Complete Task Performance

2. Direct Manipulation Interface

The basic structure of the Direct Manipulation Interface includes five windows as shown in Figure 5: (1) a Directory Window which displays the hierarchial directory structure, (2) a File Window which displays the files listed in a selected directory, (3) a File Sort Window which displays the files of the selected directory sorted by name, date of creation, or size, as designated by the user, (4) a New Name Window which displays all the names available for the create a file/directory and rename a file commands, and (5) an Icon Window which contains all the icons used for task operations. If any of the information window views are obstructed, the

user must perform a scrolling operation by clicking and dragging with the mouse as described in the instruction set.

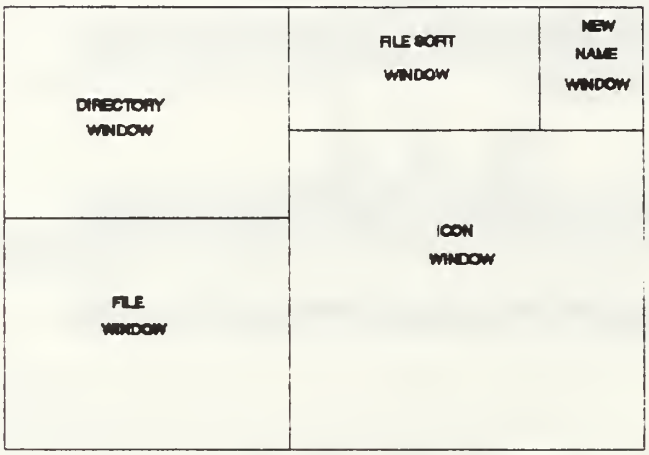


Figure 5. Direct Manipulation Window Format

The primary method for input to this interface was the mouse. The only task which required the use of the keyboard was task five, in which the user was required to type in a corrected path name when copying a file from one directory to another. All other input was performed by clicking with either the right or left mouse button as prescribed in detail in the instruction set.

When the user performs an operation properly on a file or directory, a prompter window appears allowing the user to accept or cancel the operation. If insufficient

information is provided for an operation, or an error is made in the selected information, an error message will appear containing a simple error statement as shown in Figure 6.

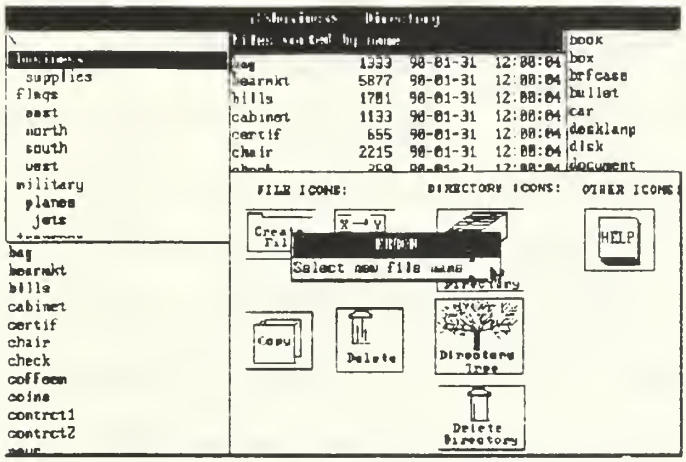


Figure 6. Direct Manipulation Screen Containing Error Message

As in the Command Language Interface, the user may access a general help screen when needed. An example screen is shown at Figure 7.

The user selects the name of the command or item that he needs help with by placing the cursor over that name and pressing the left mouse button. The right side of the Help window then displays the step-by-step instructions for the specific operation.

An example of a screen in which the user has selected the 'business' directory is provided at Figure 8. This portrays the same information as provided in Figure 4 using the Command Language Interface.

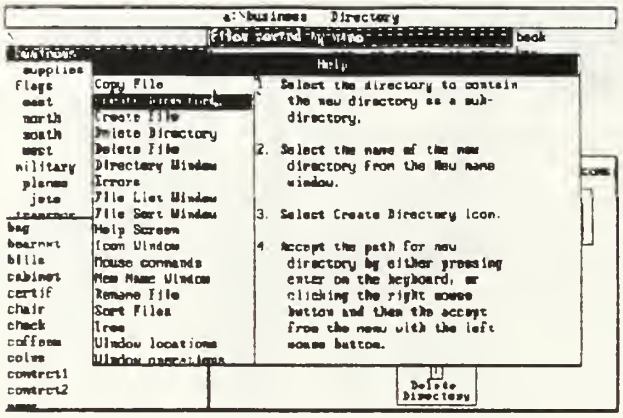


Figure 7. Direct Manipulation Help Screen - Create Directory Help

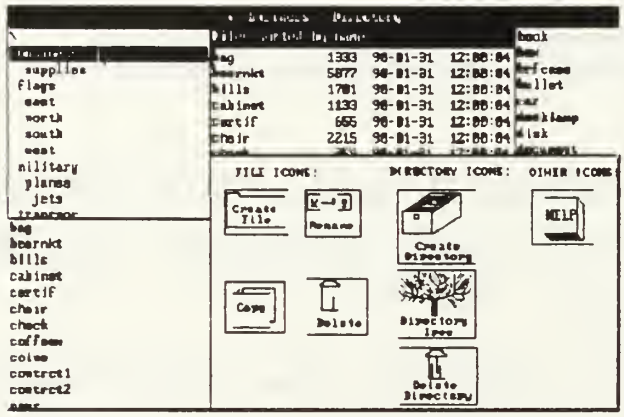


Figure 8. Direct Manipulation Screen - Complete Task Performance

B. EXPERIMENTAL DESIGN

The experiment design was developed to support the evaluation of the cognitive complexity mental processes of the typical user while performing commonly used, complex tasks involving the file management system on two separate interfaces. Five complex file management tasks were developed to test the mental algorithms required to operate within this system. A separate group of four participants was formed to perform the file management tasks on each interface. Because an important research area was the user's task representation, it was not necessary to test the same group on both interfaces. The task knowledge gained by the subject during the initial task performance on an interface nullified the ability to reevaluate him on a second interface with the same or similar tasks. Therefore, two groups of participants were used.

As each participant began the experiment, he/she was given a brief overview of the experiment and was the provided a set of instructions to read regarding the specific interface that he would be using. Each of the instruction sets contained parallel information--a brief overview of operating and file management systems in general, a detailed description of the interface itself, and an introduction to the think-aloud procedure that would be utilized in the experiment. Copies of each of the instruction sets are at Appendices A and

B. It was emphasized throughout this introductory phase that it was the interface, and not the individual, that was being evaluated. Once the participant completed reading the instructions, he was given the five tasks to perform, told that time was not a factor of this evaluation, and asked to work through the tasks to completion.

The users were not given any practice session to become familiar with the interface prior to the start of the evaluation. During preliminary test runs of the experiment tasks, it was observed that much learning was acquired during the practice session and that many of the user's vital concerns, misperceptions, and errors were lost for analysis during this practice session. Therefore, the practice session, as well as a set of simple tasks, were removed from the experimental process. This was not done without known consequences--the user's work through of the early tasks was particularly cumbersome as it was necessary for him to undergo a very steep learning curve in order to successfully complete the task. This lack of familiarity forced the users to use the help screens extensively. There were also high rates of error in learning syntax procedures for the Command Language interface and mouse techniques for the Direct Manipulation interface. It also became necessary for the observer to intervene, on occasion, to assist the user in order for him to proceed with the task at hand.

The observer's role was vital to the think-aloud protocol procedure. Each experiment was conducted individually in order for the observer to be present to ensure that the "talking aloud" continued throughout the experiment. Although the observer was stationed along side of the participant, the observer attempted to be as non-obtrusive as possible. The role of the observer was to observe, and not to interview, the participant. The observer prompted the user to continue to verbalize his actions, thoughts, and concerns throughout the experiment. The observer only provided help if the participant was unable to continue working through the task because of a repeated error or if, after trying all means of obtaining assistance from the interface, he could not take another step towards accomplishing the task.

C. SELECTED TASKS

Five rudimentary tasks, which required complex cognitive thinking--such as establishing a goal hierarchy, operators, methodology, and selection rules--were designed for the experiment. The selection insured that the users were required to perform each of the commands available within the interfaces and that the users demonstrate an understanding of the relationships of the directories, subdirectories, and files within the management system. The five tasks are provided below, along with an explanation of the minimal

cognitive requirements of each. The capitalized tasks are stated just as they were written for the participants to perform.

TASK 1: FIND THE FILE CALLED *PLANE* AND COPY IT TO A FILE CALLED *AIRCRAFT* WITHIN THE SAME DIRECTORY

The file called *plane* is located within the subdirectory *airtrans*. Therefore, the first goal of the user is to locate the file--because it was similarly named with the subdirectory *planes*, the user was forced to distinguish the name differential, as well as the file-subdirectory differentiation. Once located, the user must then select the copy operations to copy the file back into the correct subdirectory.

TASK 2: CREATE A FILE CALLED *CAR* IN THE *GROUND* DIRECTORY AND SORT *GROUND* FILES BY FILE SIZE

The user's first goal is to locate the subdirectory *ground*. Once this is completed, the goal is to create a file named *car* and put it in this directory. This involves selecting the proper operators to create the file and selecting the correct path name for its

location. The user must then select and execute the operators required to sort the files by size.

TASK 3: DELETE THE *PLANES* DIRECTORY

In order to delete the *planes* directory, the user must first locate the subdirectory and then determine what the subdirectory contains. The user must then remove all the contents of the directory prior to attempting to delete the file. The user must, therefore, identify the *planes* subdirectory called *jets* and subsequently remove the contents of it, as well as the *jets* subdirectory itself. Once removing all of the contents of *planes*, the user many then remove the directory *planes*.

TASK 4: FIND THE LARGEST FILE OF ALL THE DIRECTORIES AND RENAME THE FILE TO *LARGE.FIL*

The initial goal of this task is to locate the largest file. This requires the users to sort each of the directories and subdirectories by size in order to locate the largest of all files. Included in this is the requirement to compare the file sizes of those files located at the directory level. Once all of the file sizes have been compared and the largest file located, the user's goal is to rename the file *large.fil*. The

user must then select the correct operators and execute the command to rename a file.

TASK 5: THE SYSTEM SUPERVISOR INFORMS US THAT THE *GROUND* DIRECTORY IS A MISNOMER AND SHOULD REALLY BE CALLED THE *FLEET* DIRECTORY. ALSO, HAVING A *GROUND* DIRECTORY CAUSES CONFUSION AMONG THE STAFF. RECTIFY THE SITUATION.

The user's first goal is to create a directory called *fleet* and place it in the same directory as *ground*. The user must, therefore, locate the *ground* directory and note its higher level directory. The user must then select and complete the commands required to create a new directory. Once the new directory is created within the higher level directory of *ground*, the user must then move the files from *ground* to *fleet* and remove the *ground* directory. To move the files, the user must copy the files individually from *ground* to *fleet* and the remove each file from *ground*. After all files have been moved, the *ground* directory must then be removed.

D. PARTICIPANTS

The type participants chosen for the experiment were intended to emulate the intended user of most interfaces--that of an inexperienced novice. Eight students participated in the experiment. Unlike more typical data oriented

experimental protocols where larger numbers of participants are needed to make certain of the accuracy of the outcomes, verbal protocol analysis requires only a few participants. The reasons are twofold: 1) the wealth of information obtained from such a protocol requires a tremendous effort of compilation and analysis, such that large numbers of participants are not feasible and 2) each participant's verbalized representation of the task performances are extremely insightful and useful at pinpointing problems. The eight volunteers, four men and four women, for the experiment are all graduate students, recently enrolled at the Naval Postgraduate School. All had little or no exposure to computers and operating file management systems. The age range of the students was 24 to 40, with a mean age of 32.7. As each student was a graduate level student, their verbalization skills were judged to be above average. The eight were randomly assigned to each interface; although it was ensured that an equal number of men and women were assigned to each interface.

E. EXPERIMENTAL ENVIRONMENT

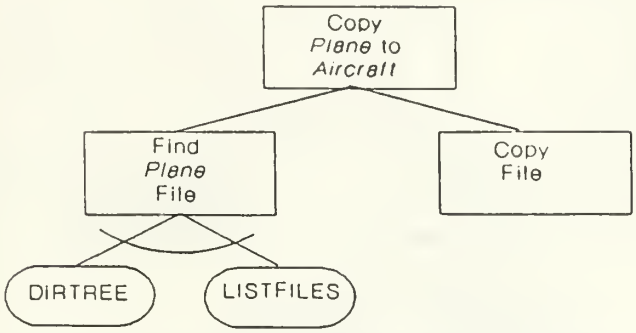
Each of the experiments was conducted on an individual basis and performed in a quiet, laboratory setting that was not unlike most office environments. The subjects were each provided with a A UNISYS 8386 computer, monitor, and keyboard.

Participants utilizing the Direct Manipulation interface were also provided with a mouse. The users were allowed usage of a pencil and paper to take notes during the experiment, if they so desired. The experiment was recorded on an ordinary tape recorder which contained a built-in microphone. The tape recorder was placed conspicuously near the monitor, so as not to intimidate the participant. Because the microphone was very sensitive to noise pick-up, the user could speak comfortably with a normal to low voice range. The observer sat to the side of the participant throughout the experiment, interfering as little as possible.

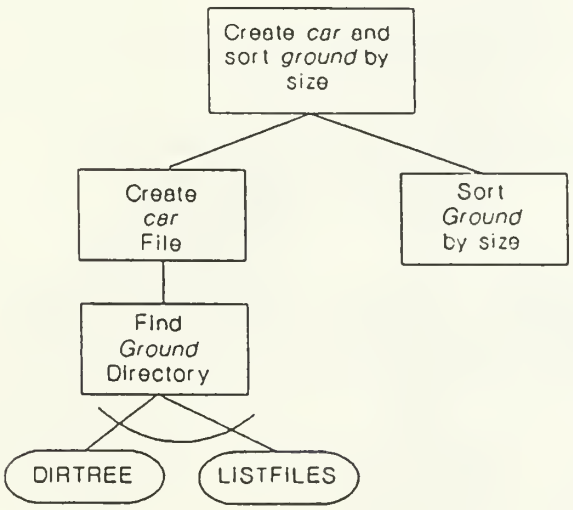
F. ANALYSIS OF DATA

Prior to the start of actual conduct of the experiment, baseline goal hierarchies and production rule sets, in the form of chronological flow charts, were generated for each task within each interface. These are provided in the below figures.

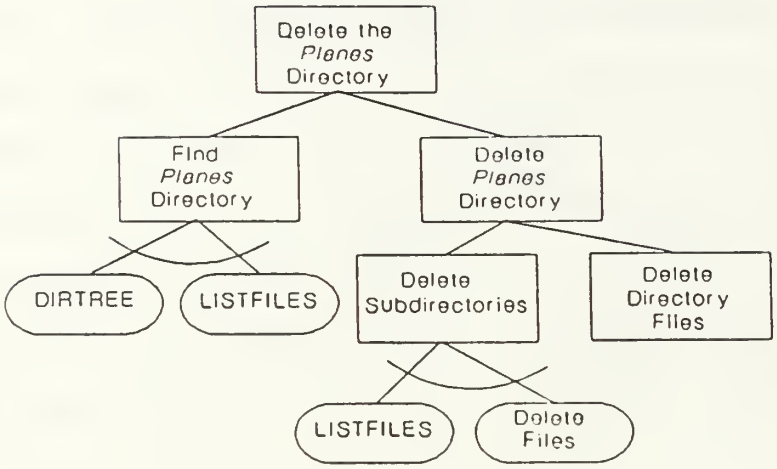
Find the file called *.plane* and copy it to a file called *aircraft* within the same directory.



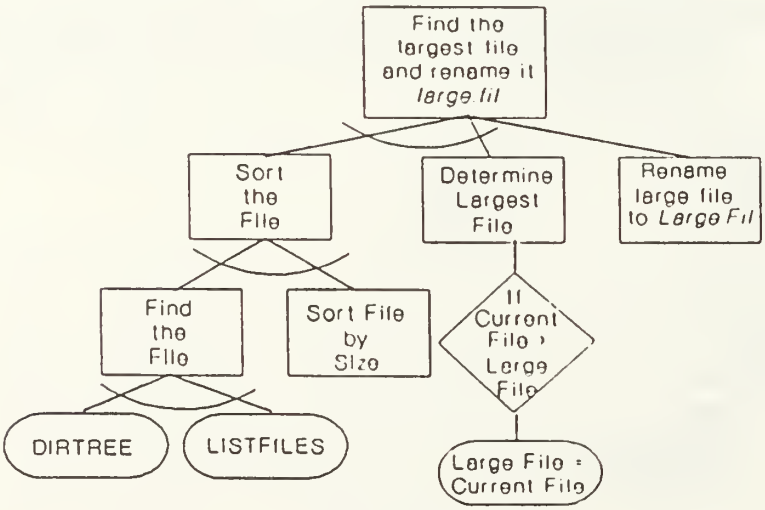
Create a file called *car* in the *ground* directory and sort *ground* files by file size.



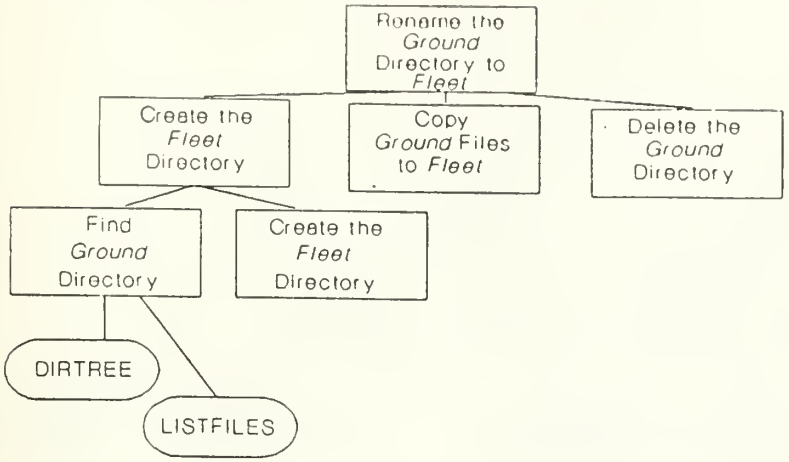
Delete the *planes* directory.



Find the largest file of <u>all</u> the directories and rename the file to *large.fil*.

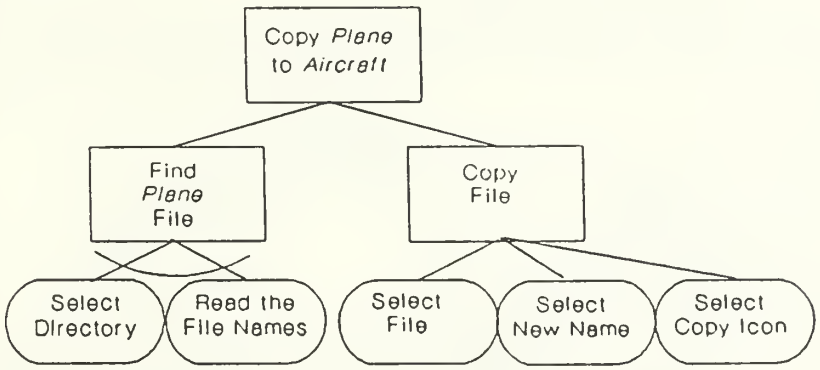


The system supervisor informs us that the *ground* directory is a misnomer and should really be called *fleet* directory. Also, having a *ground* directory in the system causes confusion among the staff. Rectify the situation.

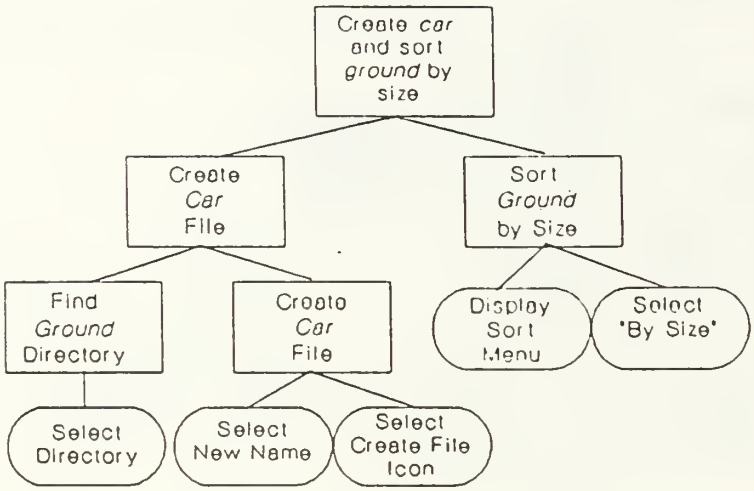


Direct Manipulation Interface Goal Hierarchies

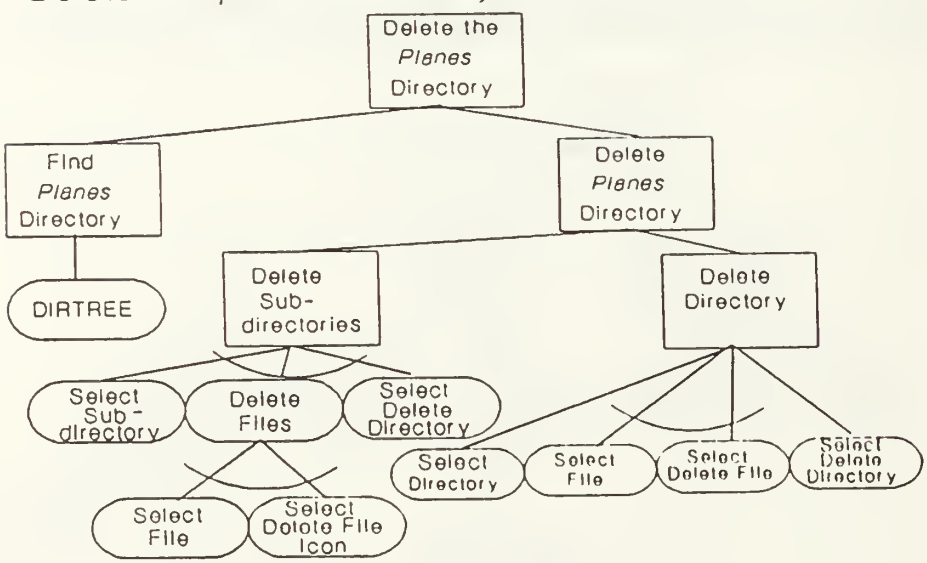
Find the file called *plane* and copy it to a file called *aircraft* within the same directory.



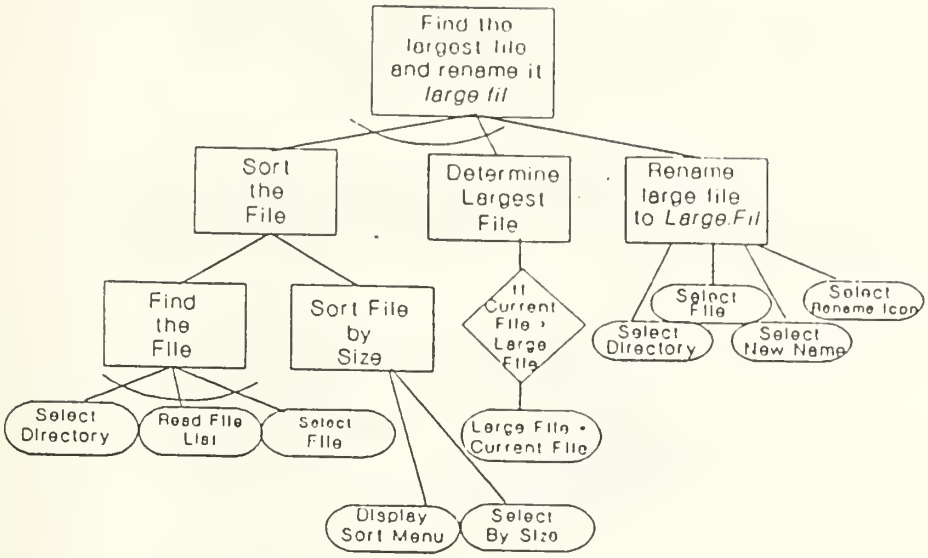
Create a file called *car* in the *ground* directory and sort ground *files* by file size.



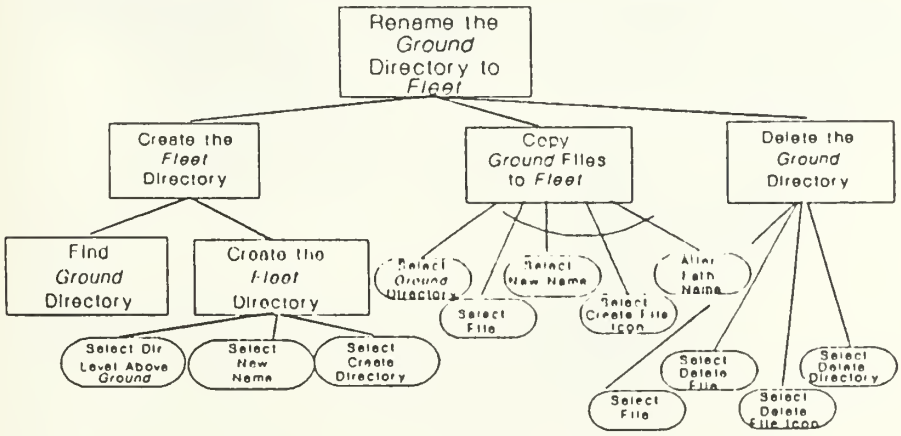
Delete the *planes* directory.



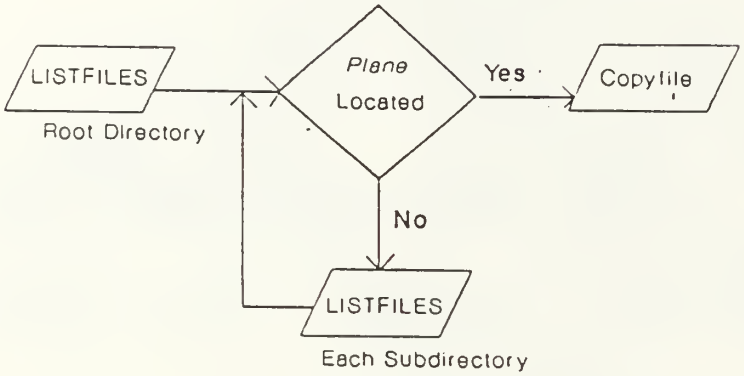
Find the largest file of all the directories and rename the file to *large.fil*.



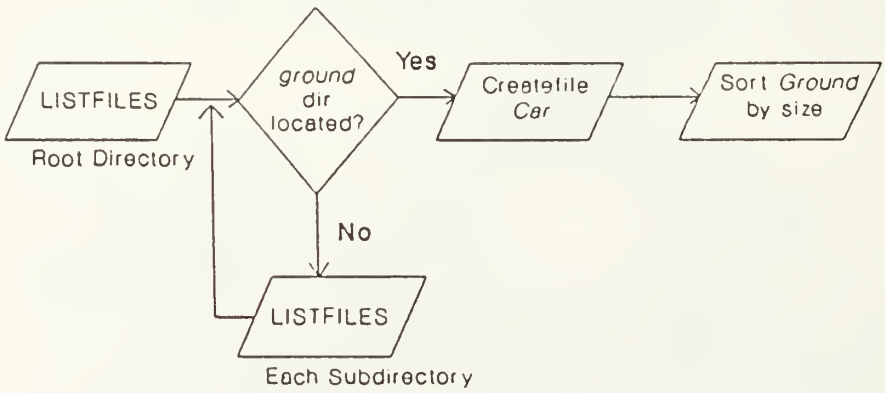
The system supervisor informs us that the *ground* directory is a misnomer and should really be called *fleet* directory. Also, having a *ground* directory in the system causes confusion among the staff. Rectify the situation.



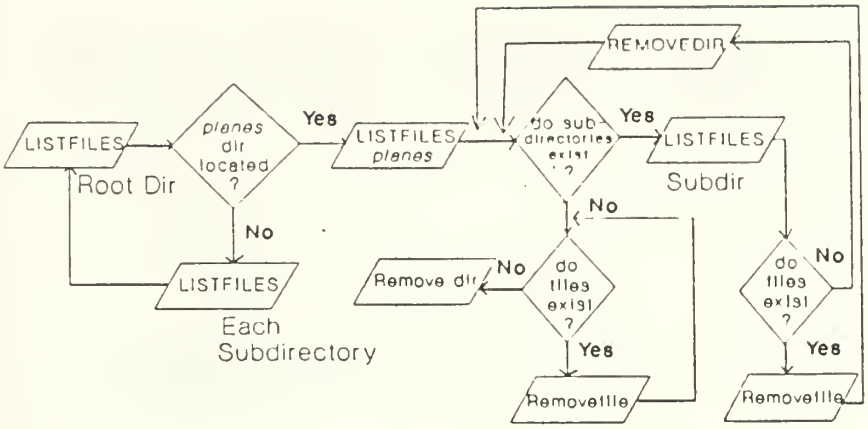
Find the file called *plane* and copy it to a file called *aircraft* within the same directory.



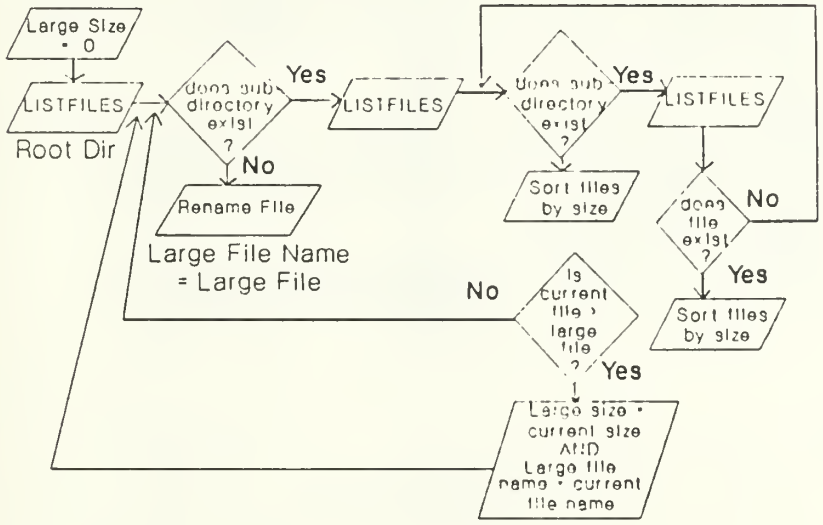
Create a file called *car* in the *ground* directory and sort *ground* files by file size.



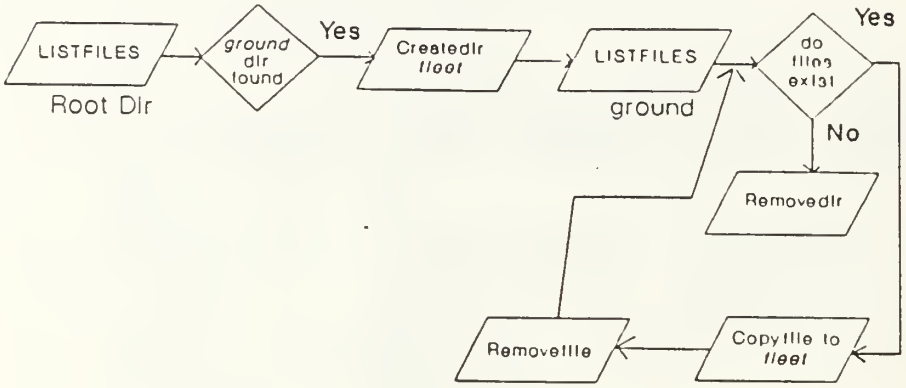
Delete the *planes* directory.



Find the largest file of <u>all</u> the directories and rename the file to *large.fil*.

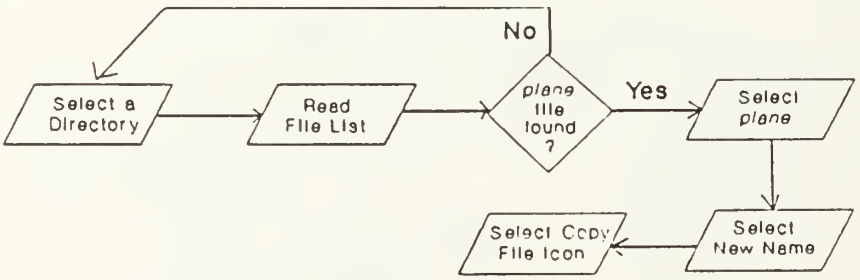


The system supervisor informs us that the *ground* directory is a misnomer and should really be called *fleet* directory. Also, having a *ground* directory in the system causes confusion among the staff. Rectify the situation.

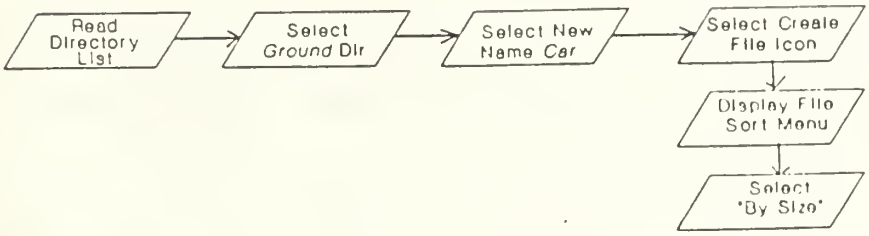


Direct Manipulation Interface Flowcharts

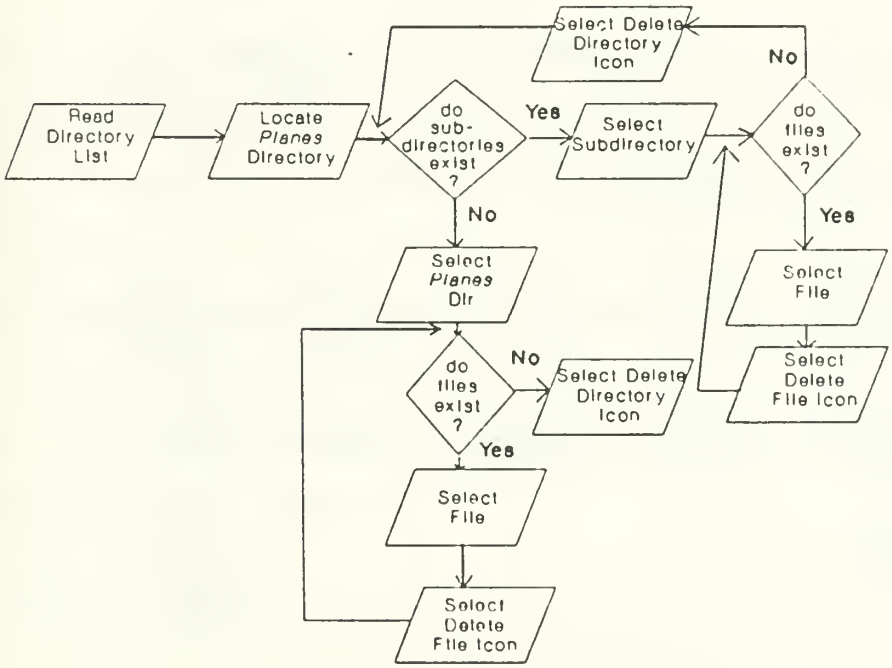
Find the file called *plane* and copy it to a file called *aircraft* within the same directory.



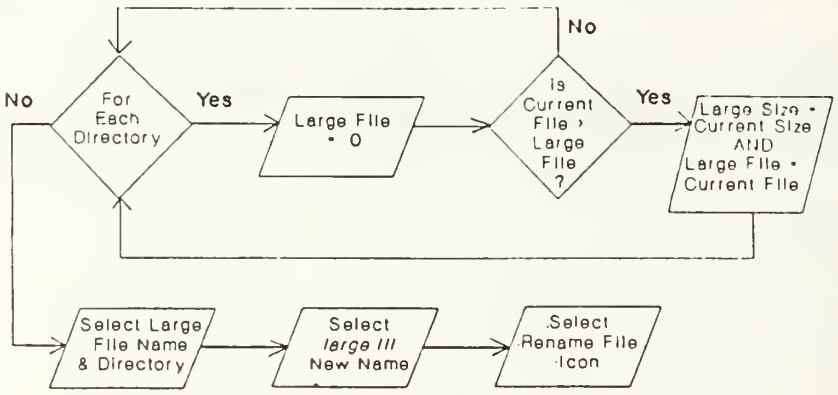
Create a file called *car* in the *ground* directory and sort ground *files* by file size.



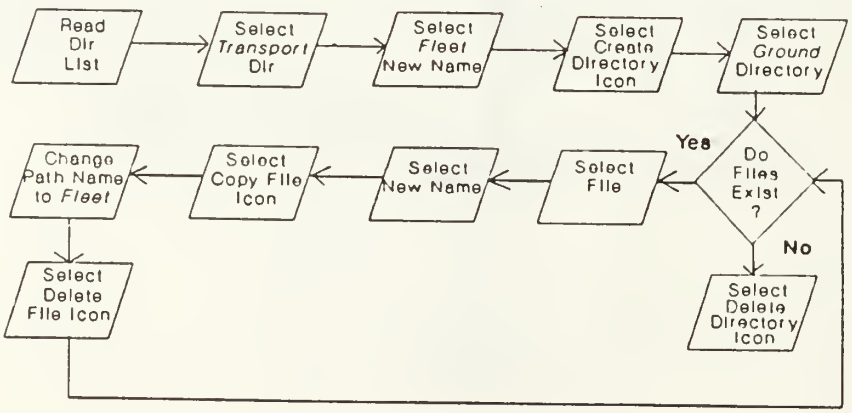
Delete the *planes* directory.



Find the largest file of all the directories and rename the file to *large.fil*.



The system supervisor informs us that the *ground* directory is a misnomer and should really be called *fleet* directory. Also, having a *ground* directory in the system causes confusion among the staff. Rectify the situation.



The analysis of the data began at the conclusion of each experiment session. The tape recording and a user log of the session was used to transcribe the data in a verbatim manner. Several more iterations of the tape transcribing resulted in a summarized version in which pertinent information was retained in verbatim form for further analysis. Based upon these transcriptions, goal hierarchies and flow charts for each individual participant's performance on a given task were developed. The user's goal hierarchies were then mapped to the baseline goal hierarchies to obtain the needed task-to-device mappings. The flow charts were similarly compared to their respective baseline flow charts and differences were computed. The following specific data elements were measured: the number of production rules, the complexity of the rules, the matching of steps in sequence within the task-to-device mappings, and, finally, the number of superfluous steps performed by each user.

To illustrate this process, the complete analysis of the task requiring the user to delete the *planes* directory will be shown as performed by one user on each of the two interfaces. This includes the summarized transcription of the experiment session, the task-to-device mapping and the flow chart generated for each user.

COMMAND LANGUAGE INTERFACE
SUMMARIZED TRANSCRIPTION
DELETE THE *PLANES* DIRECTORY

-HELP DELETEFILE; HELP; "They trick you on that one by not having it follow the same name";

-HELP REMOVEDIR; HELP REMOVEFILE; attempts to remove the *plane* directory--"Aah, it's not empty, do you want me to do it anyway? Well then, I guess it must have files in it that I have to remove first";

-HELP LISTFILE; HELP REMOVEDIR; lists the files in *planes*; removes the files in *planes*--attempts to delete the directory *jets* as if it were a file, receives error;

-"We did not previously remove *jets* . . .*jets* is a directory; well, you have to delete the files in a directory; do you have to remove directories too?";

-HELP; HELP REMOVEDIR; attempts to remove the directory--receives the error that *jets* is not empty;

-Removes all the files from *jets*; REMOVEDIR *military\planes\jets*; REMOVEDIR *military\planes*;

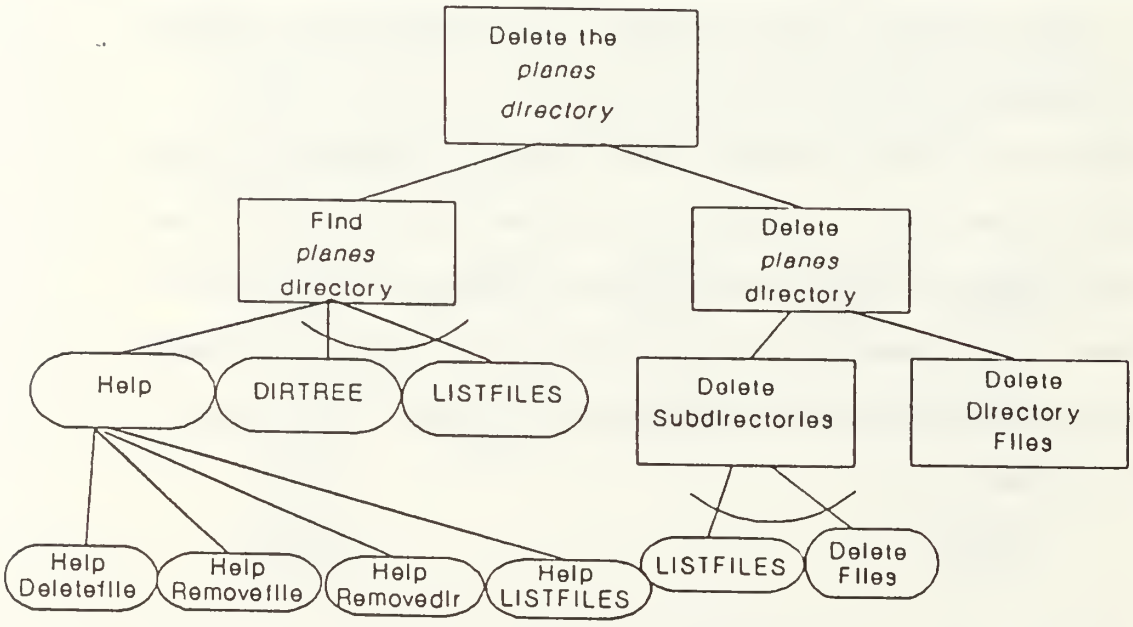


Figure 9. User's Task 3 Goal Hierarchy - Command Language

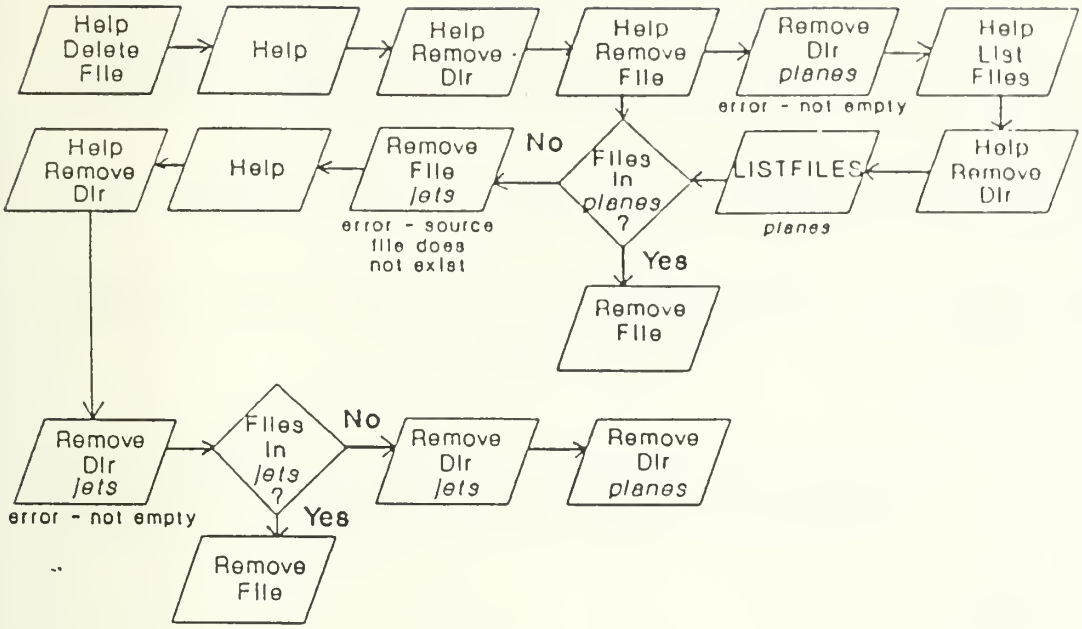


Figure 10. User's Task 3 Flow Chart - Command Language

DIRECT MANIPULATION INTERFACE
SUMMARIZED TRANSCRIPTION
DELETE THE *PLANES* DIRECTORY

-"I'm going to go over and highlight *planes* subdirectory and DELETE FILE icon; an error came up that I didn't select a file, so I guess I'm going to go over and try to select both files and see if that works";

-"Oops, it won't let me, so I'll try one at a time"; delete both files; "there are no files in my directory so I'll go ahead and try to delete the directory *planes*"; selects the DELETE FILE icon and receives same error;

-"I'm going back up and try to get a menu from *planes*, can't get one; oh, I see the error . . .there is a DELETE DIRECTORY icon which I should be using"; selects the DELETE DIRECTORY icon--receives an error that the directory is not empty;

-"I have no idea what that means so I guess I better look at the HELP; I'm going down to DELETE DIRECTORY, maybe that's it because *planes* is a subdirectory"; reads HELP window and discovers that all subdirectories must be removed in order to remove the directory;

-Deletes the files under *jets*; "I don't know whether *jets* is a file or a subdirectory so I'll try to delete the file--nope, o.k., delete directory";

-"Now finally, I can go ahead and delete the *planes* directory"; deletes the *planes* directory;

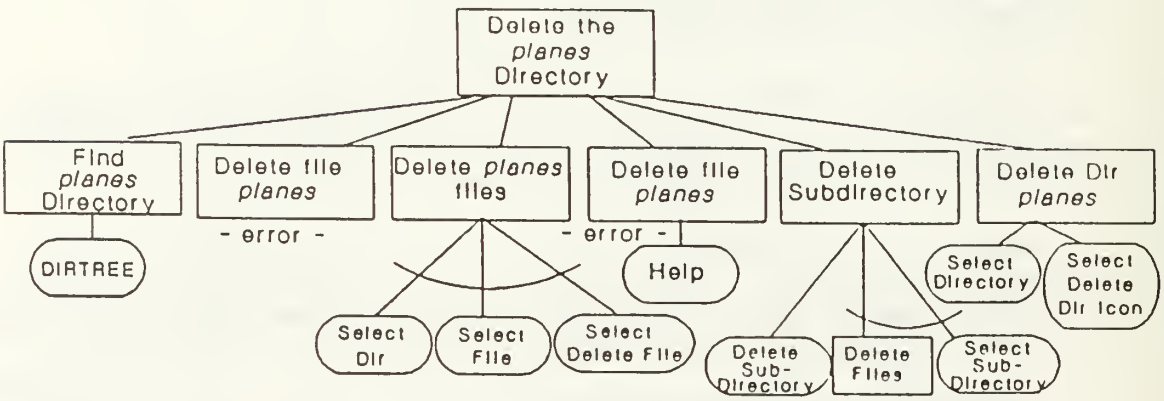


Figure 11. User's Task 3 Goal Hierarchy - Direct Manipulation

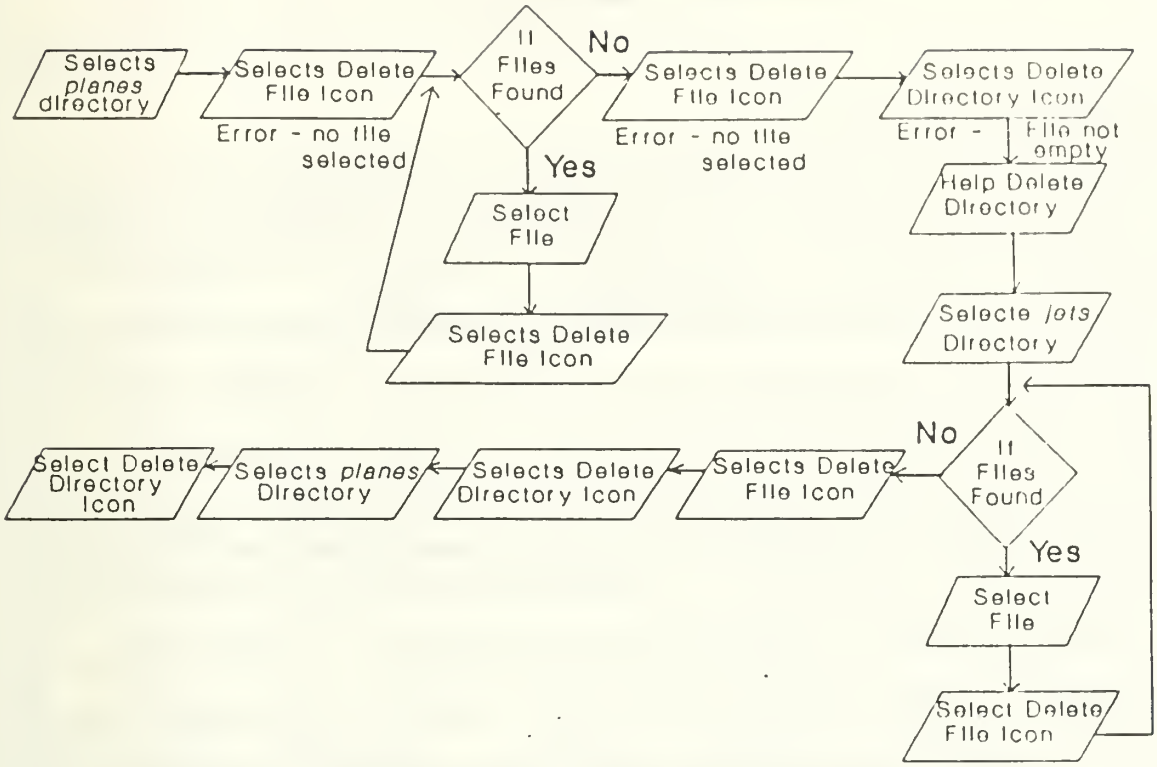


Figure 12. User's Task 3 Flow Chart - Direct Manipulation

IV. DATA ANALYSIS

A. METHOD/SAMPLE SIZE

The data extracted from the goal hierarchies and flow charts were utilized for comparisons between two groups--the Command Language and Direct Manipulation Interfaces. The data was computed by comparing 20 (4 subjects, 5 tasks) data points: each of the four users within a group performed five tasks. Four dependent variables were computed--the number of production rules generated, the complexity of the production rules, task-to-device mapping, and the number of unnecessary steps added to the production system. While the number of production rules generated by the user and the number of extraneous steps were counted, the relative complexity of the production rules and the strength of the task-to-device mapping is a relative rating score. Each was compared to the baseline data and was rated between zero and one, with the rating of zero implying strict adherence to the baseline model. The mean and standard deviation of each measure was calculated and a series of one-tailed t-tests were performed on each measure.

B. RESULTS

A table which summarizes the analyzed data is provided at Table 2.

TABLE 2: SUMMARY OF DATA ANALYSIS				
VARIABLE	MINIMUM VALUE	MAXIMUM VALUATION	MEAN	STANDARD DEVIATION
Number of Production Rules				
CLI	3.0	6.0	3.9	.91
DMI	1.0	5.0	2.55	1.05
Confidence Interval (t (38) = 4.3.411, P < .0001)				
Complexity of Production Rules				
CLI	0	.98	.48	.26
DMI	0	.8	.31	.21
Confidence Interval (t (38) = 2.0781, P < .005)				
Degree of Mapping Correlation				
CLI	.43	1.0	.63	.18
DMI	0	.7	.4	.17
Confidence Interval (t (38) = 4.2589, P < .0001)				
Unnecessary Actions				
CLI	2.0	6.0	4.15	1.31
DMI	0	5.0	2.3	1.34
Confidence Interval (t (38) = 4.4141, P < .0001)				

Analysis of the four specifically measured areas produced the following results:

1. **Number of Production Rules**

Users of the Command Language Interface generated more production rules than Direct Manipulation users, with means of 3.9 and 2.55, respectively.

2. **Complexity of Rules**

On a scale of zero to one, Command Language users created more complex production rules than Direct Manipulation users.

3. **Task-to-device Mapping**

There was a much stronger task-to-device mapping for Direct Manipulation interface users.

4. **Number of Unnecessary Steps**

Command Language users performed an average of nearly two more unnecessary steps in the task performances than did the Direct Manipulation users.

C. IMPLICATIONS OF RESULTS

The results of this experiment have significant implications for four major areas of user interface development--how the nature of the interface affects the user's mental model, the performance of users operating on an interface, the systems design process, and implications for the "think aloud" methodology. Each of these will be addressed.

1. Affects on Mental Models

User interfaces do affect the mental models of users. The design and development of the interface can 1) shift the focus of the user's attention to various aspects of the operating requirements, 2) determine the degree of working memory load, and 3) determine the user's ability to recover from errors. The results of these three affects on mental models are a more simplified mental model for the user.

Specifically, by choosing a Direct Manipulation language, the designer will allow the user to focus on the task at hand--as he will be able to visualize the object of interest as well as the actions (via the icons) that are required to be performed on those objects. The user does not have to learn or memorize the syntax required of each command --the command icons are before him at all times and may be accessed by the mere pressing of a button on a mouse. The user may therefore concentrate his efforts on the elements required of each task--the interface has become invisible to him.

On the other hand, the Command Language user must know the commands prior to typing them in for execution. Knowing the commands initially requires the user to memorize commands and, ultimately, requires the user to learn the commands through repetitive recognize-act cycles prior to the command becoming a part of long term memory. While users of

the Direct Manipulation interface must ensure that all of the required information is selected prior to being successfully executed, he may act intuitively by comparing the objects of the task and the windows available to him--he must not memorize any commands. This requires the user to focus on the interface itself, as well as the task to be performed. This, of course, increases the load on the working memory of the Command Language user. A second aspect of the interface can provide alternatives for reducing the burden of requirements on working memory. The constant visualization of data such as the directory window in the Direct Manipulation interface reduces the load on working memory significantly, which also effects the user's ability to detect the cause of and to recover from errors. The visual representations of the Direct Manipulation and the reduction of errors extraneous to the task at hand allow the user to more directly determine and recover from errors made.

2. **Affects of User Performance**

The implication of a more simple user model has direct effects on the performance of the user--particularly in the case of the infrequent or novice user. This results in increased learnability and ease of use. This streamlined mental model is a direct result of an aligned task-to-device mapping from the user's perspective, a reduced burden on the working memory that maintains the load within the user's

capability, and the reduced number of recognize-act requirements leading to production rule firings. The aligned goal hierarchies of the user and the task result in marked improvement in the following way. The goal hierarchies result in knowledge being compiled into groupings which relate to one another. As the user learns which actions are related to others, the user compiles the production rules into one larger production rule. When there is a high correlation between tasks--i.e., the tasks have similar goal hierarchies--there is a transfer of knowledge from one task to another [Ref. 12:pp 195]. Similarly, user performance is improved by reducing the load on working memory capacity. Working memory does have limitations in that it relies on the strength of the production rules in them. These rules acquire strength through successive, successful application of the rule. If the rule is weak or incorrect, the recognize-act cycle will fire incorrect production rules. All of this can lead to the following working memory failures; loss of declarative knowledge, loss of a goal, or loss of a discriminating feature of the production rule [Ref. 12:pp 203]. Incorrect or unnecessary production rules generated by the user can further negatively impact the user's performance in that the user must discriminate which of the production rules are the correct ones--having multiple, uncompiled or weak production rules within memory can only result in increased possibilities of

incorrect actions by the user and slower learning. Thus, a more streamlined, simplified mental model may have significant impacts on increased user performance.

3. Affects of Systems Design

Using the think-aloud protocol, in conjunction with the formalized production rule framework as prescribed by Kieras and Polson, provides the system designer with a new tool to focus on in the design process. The usage of framework in this context may allow the designer to shift his focus early in the design phase to a thorough task analysis. The early focus on empirical feedback from the users and their perception of the device/task allows the designer to more accurately specify the knowledge requirements prior to the development of the system. Prototyping can begin at the earliest phases of design as the designer can use paper mock-ups of the proposed interface to observe users in the think-aloud protocol and receive important input to achieving a strong task-to-device mapping in the final product. This lends itself well for the designer to achieve an iterative design approach, while avoiding costly modifications to the system. Additionally, the pinpointed problem area feedback provided by this protocol provides a detailed evaluation of the relative complexity of alternative designs--thus allowing the designer to identify the trade-offs of design issues. Based upon this type information, the designer may then be

able to design better documentation and training programs which account for the trade-offs resulting from the final design decisions [Ref. 14:pp 300].

4. Affects on Think-Aloud Methodology

The results of this study further substantiate the effectiveness of the think-aloud protocol as a useful evaluation technique for the designer. Furthermore, it is particularly effective for use in conjunction with the production system model for the following reasons. It supports the development of production rules as it is the only currently used methodology which provides a "genuine" user's perspective of the user's task and device representations. Both the goal hierarchies and the user's "how-it-works" knowledge are more precisely defined. Thus, the task-to-device representation becomes more accurate as opposed to the designer's inferences of what the user was attempting to do. According to Wright and Monk's 1991 study, designers are poor at predicting exact problems which will surface in their own system design [Ref. 13:pp 56]. It is therefore important that the user's feedback be utilized because the baseline production rule system predicted little significant difference in the predicted cognitive complexity of the Command Language and Direct Manipulation interfaces. And finally, this methodology allows for a very small sample size--even one user--to provide immediate and effective feedback--thus

allowing for minimal cost, in terms of both time and dollars, evaluations which can be performed throughout the system life cycle.

V. SUMMARY

The think-aloud protocol was used in this exploratory study to demonstrate the effectiveness of Kieras and Polson's theoretical model, GOMS and the Cognitive Complexity Model. The results of the experiment provide further quantifiable evidence that the use of this framework may be applied by system designers to develop user interfaces that are more easily learned and used.

To analyze the effectiveness of the GOMS and Cognitive Complexity Model, an experiment comparing the cognitive processes of users on two interfaces, the Command Language and Direct Manipulation Interfaces, was conducted. Two groups of users performed a set of five complex file management tasks on their respectively assigned interfaces. As each user performed the required tasks, the session was tape-recorded to capture the verbalized thought processes of the users as they "thought-aloud" through the execution of the task. Upon completion of the eight tape-recorded sessions, each of the tapes were transcribed and analyzed to develop both a goal hierarchy and a production rule set for each task performed by an individual user. A task-to-device mapping was performed to determine the alignment of the users' perspective of the task representation with that of the designer. Each of the mapping

sets for each task were then compared across interfaces to determine which interface had better correlated task-to-device mappings. Further, the production rule sets of each interface were compared, by task against the baseline production rule sets, to determine the relative complexity of each of the interfaces.

The think-aloud process was chosen as the methodology for conducting this experiment because of its insights of the users perceptions of both the task and device representations. Current evaluation techniques used in quantifying complexity based on production rules rely on execution times and by identifying the number and type of errors committed by the user. This protocol provides detailed observations which, not only pinpoint problem areas but, delineate why the problems are occurring. This added why information strengthens Kieras and Polson's theory of cognitive complexity by providing more accurate goal hierarchies and task-to-device mappings.

Specifically, the experimental results provide four distinct implications for insight into the study of cognitive processes are they are related to human-computer interaction. First of all, the nature of the interface design influences the users' mental models of a system. Secondly, the complexity of the resulting mental model has a direct affect on the user's performance on the given interface. This can be measured by the number of production rules developed by the

user in task performance, the complexity of the production rules generated, the completeness of the task-to-device mapping, and the number of superfluous steps generated by the user. Users with more complex mental models of the system create greater demands on working memory, leading to an increased chance of error. The third implication of this study is that the use of a formalized production rule system, used in conjunction with the think-aloud protocol, may alter and improve the design process of the user interfaces by giving the designer specific feedback on design alternatives early in the design process. Finally, the study lends further support to the think-aloud protocol as a valid, effective tool in assessing the cognitive process of human-computer interaction.

LIST OF REFERENCES

1. Coventry, Lynn, "Some Effects of Cognitive Style on Learning UNIX," *International Journal of Man-Machine Interface*, Vol 31, No 7, 1989, pp 349-365.

2. Kieras, David E. and Polson, Peter, "An Approach to the Formal Analysis of User Complexity," *International Journal of Man-Machine Interface*, Vol 22, No 4, April 1985, pp 365-394.

3. Margono, S. and Shneiderman, Ben, "A Study of File Manipulation by Novices Using Commands vs. Direct Manipulation," *26th Annual Technical Symposium*, June 1987, pp 154-159.

4. Karat, J., Fowler, R., and Gravelle, M., "Evaluating User Interface Complexity," *Human Computer Interaction - INTERACT '87*, 1987, pp 489-495.

5. Te'eni, Dov, "Direct Manipulation as a Source of Cognitive Feedback: A Human-Computer Experiment with a Judgement Task," *International Journal of Man-Machine Interface*, July 1990, pp 1-25.

6. Lewis, E., "Using the 'Thinking Aloud' Method in Cognitive Interface Design," Lecture 28, Ann Arbor, Michigan: The University of Michigan Chrysler Center for Continuing Engineering Education, 1982, pp 1-16.

7. Card, Stuart K., Moran, Thomas P., and Newell, Allen, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Publishers, 1983.

8. Kieras, David E. and Bovair, Susan, "The Acquisition of Procedures from Text: A Production-System Analysis of Transfer of Training," *Journal of Memory and Language*, Vol 25, 1986, pp 507-524.

9. Polson, Peter G. and Kieras, David E., "A Quantitative Model of the Learning and Performance of Text Editing Knowledge," *CHI '85 Proceedings*, April 1985, pp 207-212.

10. Bovair, Susan, Kieras, David E., and Polson, Peter G., "The Acquisition and Performance of Text-Editing Skill:

A Cognitive Complexity Analysis," *Human Computer Interaction*, Vol 5, 1990, pp 1-48.

11. Deimel, Lionel (Editor), "User Interface Development," *Support Material for User Interface Development*, Carnegie Mellon University, SEI-CM-17-1.0, April 1988.

12. Anderson, John R., "Skill Acquisition: Compilation of Weak-Method Problem Solutions," *Psychological Review*, Vol 94, No 2, 1987, pp 192-210.

13. Wright, Peter C., and Monk, Andrew F., "The Use of Think-Aloud Evaluation Methods in Design," *SIGCHI Bulletin*, January 1991, pp 255-273.

14. Gould, John D., and Lewis, Clayton, "Designing for Usability: Key Principles and What Designers Think," *Communications of the ACM*, Vol 28, No 3, March 1985, pp 300-311.

BIBLIOGRAPHY

1. Anderson, John R., "Skill Acquisition: Compilation of Weak-Method Problem Solutions," *Psychological Review*, Vol 94, No 2, 1987.

2. Bovair, Susan, Kieras, David E., and Polson, Peter G., "The Acquisition and Performance of Text-Editing Skill: A Cognitive Complexity Analysis," *Human Computer Interaction*, Vol 5, 1990.

3. Card, Stuart K., Moran, Thomas P., and Newell, Allen, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Publishers, 1983.

4. Carroll, John M., and Mack, Robert L., "Learning to Use a Word Processor: By Doing, By Thinking, and By Knowing," *Human Factors in Computer Systems*, edited by Thomas and Schneider, Ablex, Norwood, New Jersey, 1984.

5. Carroll, John M., *Interfacing Thought*, MIT Press, Cambridge, Massachusetts, 1987.

6. Coventry, Lynn, "Some Effects of Cognitive Style on Learning UNIX," *International Journal of Man-Machine Interface*, Vol 31, No 7, 1989.

7. Deimel, Lionel (Editor), "User Interface Development," *Support Material for User Interface Development*, Carnegie Mellon University, SEI-CM-17-1.0, April 1988.

8. Ericsson, K. Anders, and Simon, Herbert A., *Protocol Analysis: Verbal Reports as Data*, The MIT Press, Cambridge, Massachusetts, 1984.

9. Gould, John D., and Lewis, Clayton, "Designing for Usability: Key Principles and What Designers Think," *Communications of the ACM*, Vol 28, No 3, March 1985.

10. Haas, Christina, "Does the Medium Make A Difference? Two Studies of Writing With Pen and Paper and with Computers," *Human-Computer Interaction*, Vol 4, No 2, 1989.

11. Jerrams-Smith, Jennifer, "An Attempt to Incorporate Expertise about Users into an Intelligent Interface for

UNIX," *International Journal for Man-Machine Interface*, Vol 31, 1989.

12. Karat, J., Fowler, R., and Gravelle, M., "Evaluating User Interface Complexity," *Human Computer Interaction - INTERACT '87*, 1987.

13. Kieras, David E., and Bovair, Susan, "The Role of a Mental Model in Learning to Operate a Device," *Cognitive Science*, Vol 8, 1984.

14. Kieras, David E., "A Model of Reader Strategy for Abstracting Main Ideas from Simple Technical Prose," *Text*, Vol 2, No 1-3, 1982.

15. Kieras, David, and Polson, Peter, "An Approach to the Formal Analysis of User Complexity," *International Journal of Man-Machine Interface*, Vol 22, No 4, April 1985.

16. Kieras, David E., and Bovair, Susan, "The Acquisition of Procedures From Text: A Production-System Analysis of Transfer of Training," *Journal of Memory and Language*, Vol 25, 1986.

17. Kitajima, Muneo, "A Formal Representation System for the Human-Computer Interaction Process," *International Journal of Man-Machine Interface*, Vol 30, No 6.

18. Lewis, E., "Using the 'Thinking Aloud' Method in Cognitive Interface Design," Lecture 28, Ann Arbor, Michigan: The University of Michigan Chrysler Center for Continuing Engineering Education, 1982.

19. Lewis, Clayton, and Gould, John D., "Designing for Usability: Key Principles and What Designers Think," *Communications of the ACM*, Vol 28, No 3, March 1985.

20. Margono, S., and Shneiderman, Ben, "A Study of File Manipulation by Novices Using Commands vs. Direct Manipulation," *26th Annual Technical Symposium*, June 1987.

21. Polson, Peter G., and Kieras, David E., "A Quantitative Model of the Learning and Performance of Text Editing Knowledge," *CHI '85 Proceedings*, April 1985.

22. Polson, Peter G., Muncher, Elizabeth, and Engelbeck, George, "A Test of a Common Elements Theory of Transfer," *CHI '86 Proceedings*, April 1986.

23. Te'eni, Dov, "Direct Manipulation as a Source of Cognitive Feedback: A Human-Computer Experiment with a Judgement Task," *International Journal of Man-Machine Interface*, July 1990.

24. Wright, Peter C., and Monk, Andrew F., "The Use of Think-Aloud Evaluation Methods in Design," *SIGCHI Bulletin*, January 1991.

APPENDIX A

COMMAND LANGUAGE INTERFACE

YOUR NAME: _____

SMC NO: _____

I. INTRODUCTION

The exercise you are about to participate in involves operating and evaluating a recently-developed computer file management system. You will be asked to read a short description of file management, followed by instructions for each file management operation. You will then perform a series of tasks that involve using the operations you have learned.

PRIVACY ACT

The information accompanying this experiment will be used for data collection and correlation purposes only. Information provided is voluntary.

II. FILE MANAGEMENT SYSTEM

A. OPERATING SYSTEM

An **operating system** is the software program that makes the hardware useable. The operating system can accomplish many functions such as communicating between the user and the computer (known as the user interface), sharing hardware among users, allowing users to share data among themselves, and many other functions.

An operating system's primary duty is to manage various files. Before we go into the details of files, it is important to understand the structure and organization of files.

B. DIRECTORIES AND FILES

A **software program** usually consists of several files. These files work together to produce the program that the user sees and interacts with on the screen. A **directory** contains all of the files used for a given program. For instance, all the files that operate the popular word processing program, WordPerfect 5.1, might be located in a directory called WP51. It is also possible (and highly recommended) to have

subdirectories within a directory to segregate your files further. For instance, in the WP51 directory you might have a subdirectory titled WORK for all the files relating to your work and one titled THESIS for all the files relating to your thesis.

Figure 1 shows the relationship of this WP51 directory to its subdirectories and files, and also its relationship with the top-level **Root** directory, often represented by a slash (\). Two other directories at the same level as WP51 (here called DOS and HG) also are shown, along with their files.

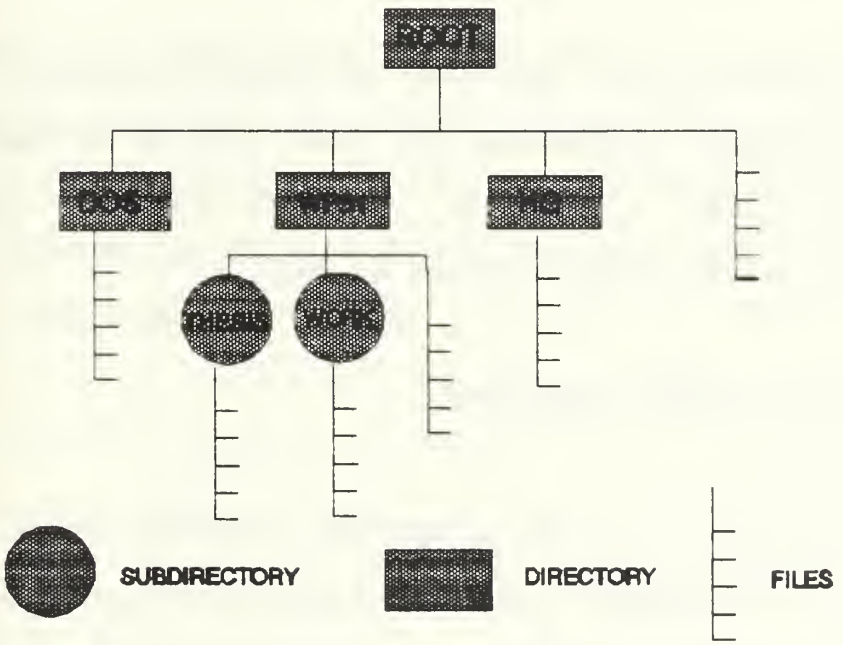


Figure 1. Relationships of the WP51 Directory with other Directories and Files.

It is often necessary to *copy* or *move* a file from one directory to another. The operating system must provide methods to move these files. For example, suppose you wrote a paper for a National Security class titled The Middle East and stored this file in the WORK directory under WP51. Later in the year you decide to use this paper in your thesis and need to transfer the file to the THESIS subdirectory. Some form of *move* or *copy* command would allow you to transfer this file. Additionally, you may want to change the file name to Chapter II. Again the operating system interface must provide a means of doing so.

As mentioned earlier, directories contain related files. The operating system interface also must provide a means to manage directories. That is, commands such as *create directory* and *delete directory* are needed.

III. OPERATING SYSTEM INTERFACE

The experimental operating system interface you will be working with is a **Command Language Interface**, Figure 2. It uses specific commands to perform an operation. The interface

contains three windows: Directory, File and Command windows.

A. Windows

1. Directory Window

The Directory Window provides a directory tree of all the subdirectories contained in a specified directory. If insufficient space is provided in the window for the entire directory tree, the user will be prompted to press M for the remainder of the tree.

2. File Window

The File Window contains a listing of all the files in a specified directory. The name of the directory appears in the label for the File Window. If insufficient space is provided in the window for all the files, the user will be prompted to press M for additional files.

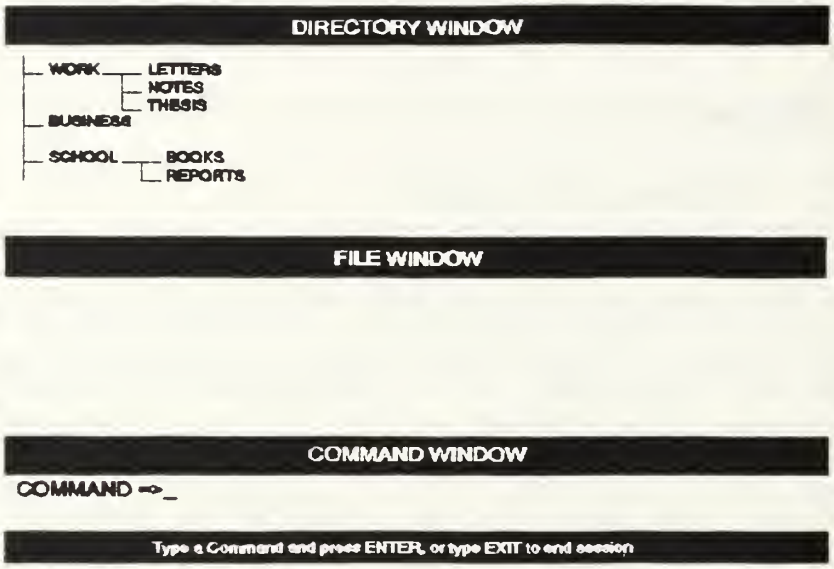


Figure 2. The Command Language Interface Windows.

3. Command Window

The Command Window allows the user to input specific commands to the interface. Each command has a specific syntax associated with it. If the command is typed incorrectly or missing arguments, an error message will appear below the command line indicating the reason for the error.

B. Commands

All commands available for the Command Language Interface are identified below. Commands must be typed in lower case, the <> indicates that the user provides a file name or directory name. The <> are not to be typed. The path for a file or directory is given using the \ and must be included in the command. If a directory is not specified, the interface will assume the root "\" directory is being referenced.

1. Copy File

A file can be copied from one directory to another using the following command:

copyfile <directory\filename> to
<directory\filename>

with the user providing the path for the existing file and new file.

2. Create Directory

A new directory can be created using the following command:

createdir <directory\new directory name>

with the user providing the path and name for the new directory.

3. *Create File*

A new file can be created using the following command:

createfile <directory\new file name>

with the user providing the path and name for the new file.

4. *Directory Tree*

A directory tree of a specified directory can be displayed on the screen using the following command:

dirtree <directory>

with the user providing the path of the directory to be displayed.

5. *Help*

The help command provides the user information regarding the interface and any command. At the command line the user can type **help** for general information or the following for more specific information about a command:

help <command>

6. *Listing Directory Files*

The files for a specific directory can be displayed using the following command:

listfiles <directory>

with the user providing the path for the directory to be displayed.

7. *Remove Directory*

When a directory does not contain files nor subdirectories, and is no longer needed, it can be deleted using the following command:

removedir <directory>

with the user providing the path of the directory to be deleted.

8. *Remove File*

A file can be deleted from a directory using the following command:

removefile <directory\filename>

with the user providing the path of the file to be deleted.

9. *Rename a File*

A file can be renamed using the following command:

renamefile <directory\filename> to <directory\new filename>

with the user providing the paths for the old and new file names.

10. Sort Files

The files in the file window can be sorted by name, date or size using the following command:

sortfile <directory> by date

sortfile <directory> by size

sortfile <directory> by name

with the user providing the path for the directory to be sorted.

IV. YOUR TASK

You are participating in an experiment that is being conducted to evaluate the interface of a recently developed computer file management system. The purpose of the experiment is to determine the strengths and weaknesses of the interface. You must perform a series of tasks using the interface. As you perform these tasks, you will be required to "think aloud"--that is, you will say aloud what you are thinking about, any questions that you may have concerning the task, or anything related to the system that may cause you confusion. Throughout this period, your thoughts will be recorded for later analysis. It is important to remember that it is not your thought process that is being evaluated, but rather the type of concerns you encounter while attempting to learn and perform tasks using the file management system provided. Although you have been asked to verbalize your questions as they arise (and these question are vital to the final evaluation of the system), it is important that you realize that your questions probably will not be answered. Finally, if you become absorbed in the performance of the task and therefore stop verbalizing your thoughts, the observer will provide prompting.

***** STOP *****

THE OBSERVER WILL ASSIST YOU IN PROCEEDING WITH THE EXPERIMENT

V. EXPERIMENT

Complete the following operations using the procedures you have read about. Use the help screen as needed. Work at a normal pace and as accurately as possible.

1. Find the file called *plane* and copy it to a file called *aircraft* within the same directory.

2. Create a file called *car* in the *ground* directory and sort *ground* files by file size.

3. Delete the *planes* directory.

4. Find the largest file of all the directories and rename the file to large.fil.

5. The system supervisor informs us that the *ground* directory is a misnomer and should really be called the *fleet* directory. Also, having a *ground* directory in the system causes confusion among the staff. Rectify the situation.

APPENDIX B

DIRECT MANIPULATION INTERFACE

YOUR NAME: _____

SMC NO: _____

I. INTRODUCTION

The exercise you are about to participate in involves operating and evaluating a recently-developed computer file management system. You will be asked to read a short description of file management, followed by instructions for each file management operation. You will then perform a series of tasks that involve using the operations you have learned.

PRIVACY ACT

The information accompanying this experiment will be used for data collection and correlation purposes only. Information provided is voluntary.

II. FILE MANAGEMENT SYSTEM

B. OPERATING SYSTEM

An operating system is the software program that makes the hardware useable. The operating system can accomplish many functions such as communicating between the user and the computer (known as the user interface), sharing hardware among users, allowing users to share data among themselves, and many other functions.

An operating system's primary duty is to manage various files. Before we go into the details of files, it is important to understand the structure and organization of files.

C. DIRECTORIES AND FILES

A **software program** usually consists of several files. These files work together to produce the program that the user sees and interacts with on the screen. A **directory** contains all of the files used for a given program. For instance, all the files that operate the popular word processing program, WordPerfect 5.1, might be located in a directory called WP51. It is also possible (and highly recommended) to have

subdirectories within a directory to segregate your files further. For instance, in the WP51 directory you might have a subdirectory titled WORK for all the files relating to your work and one titled THESIS for all the files relating to your thesis.

Figure 1 shows the relationship of this WP51 directory to its subdirectories and files, and also its relationship with the top-level **Root** directory, often represented by a slash (\). Two other directories at the same level as WP51 (here called DOS and HG) also are shown, along with their files.

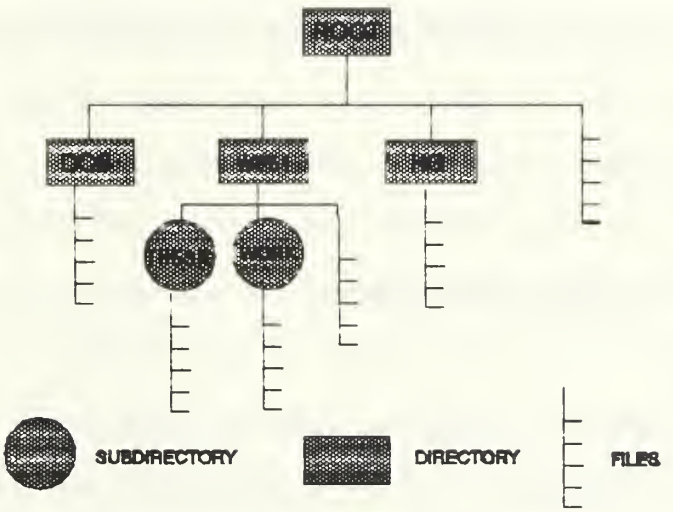


Figure 1. Relationships of the WP51 Directory with other Directories and Files.

It is often necessary to *copy* or *move* a file from one directory to another. The operating system must provide methods to move these files. For example, suppose you wrote a paper for a National Security class titled The Middle East and stored this file in the WORK directory under WP51. Later in the year you decide to use this paper in your thesis and need to transfer the file to the THESIS subdirectory. Some form of *move* or *copy* command would allow you to transfer this file. Additionally, you may want to change the file name to Chapter II. Again the operating system interface must provide a means of doing so.

As mentioned earlier, directories contain related files. The operating system interface also must provide a means to manage directories. That is, commands such as *create directory* and *delete directory* are needed.

III. OPERATING SYSTEM INTERFACE

The experimental operating system interface you will be working with is called a **Direct Manipulation Interface**. It uses the mouse device to "click" on to various icons, directory names, or file names on the screen. An icon is a graphical representation of an object or an operation. Once

activated, the icon carries out a specific function or operation, such as copying a file from one directory to another.

A. Mouse

The mouse (Figure 2) is the only input device you will be using for this experiment. The keyboard will only be operational when entering your name and SMC at the beginning of the experiment.

Hold the mouse in the right hand (if right handed, left hand if left handed) so that the cord and buttons are at the top. Place your index and middle fingers lightly over the mouse buttons. Gently guide the movement of the mouse with the hand.

Normally, the mouse is represented as an arrow, or cursor, on the screen which moves as you move the mouse. When the system is performing an operation that takes longer than one second, the cursor will appear as an hour glass to indicate that the operation will take time. The cursor is not functional in the hour glass mode.

Gently press or "click" the left button to select the item on the screen superimposed by the arrow or cursor. Click

the right button to call up a menu that describes operations you can do in a given window. Each window has a menu assigned to it. Most of the window menus are not operational. However, the help, sort, tree, and error windows do have operational menus.

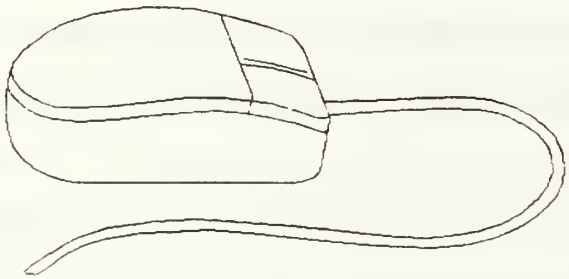


Figure 2. Mouse Control Device

B. Windows

The Direct Manipulation Interface consists of five windows (Figure 3). These are (1) Directory Window, (2) File Window, (3) File Sort Window, (4) New Name Window and (5) Icon Window. Each window has a specific function and interacts with the other windows by use of mouse operations. Additionally, the Help and Tree windows will pop up onto the

screen when the icon that represents one of them is selected.

All windows have similar characteristics. The **Top Pane** is the main window that includes all other windows and encompasses the entire screen. The label of the top pane contains the name of the interface or the name of a selected directory. The **Directory**, **File** and **New Name windows** all operate in a similar manner where the user selects an item. The selected item will appear highlighted. Due to the limited size of the windows, not all files, directories or new names may fit in the window at one time. The window operations provide the ability to **scroll** the window. Scroll a window by pressing and holding the right mouse button (cursor changes to a four directional arrow) in the window to be scrolled. Move the cursor out of the window in the direction of the additional names. The **scroll bar** to the right of the window shows the status of the scrolling with respect to the total list. The scroll bar only appears during the scrolling operation.

1. Directory Window

The Directory window contains a list of all the directories on the disk in alphabetical order. The directories are indented to reflect the hierarchical or tree

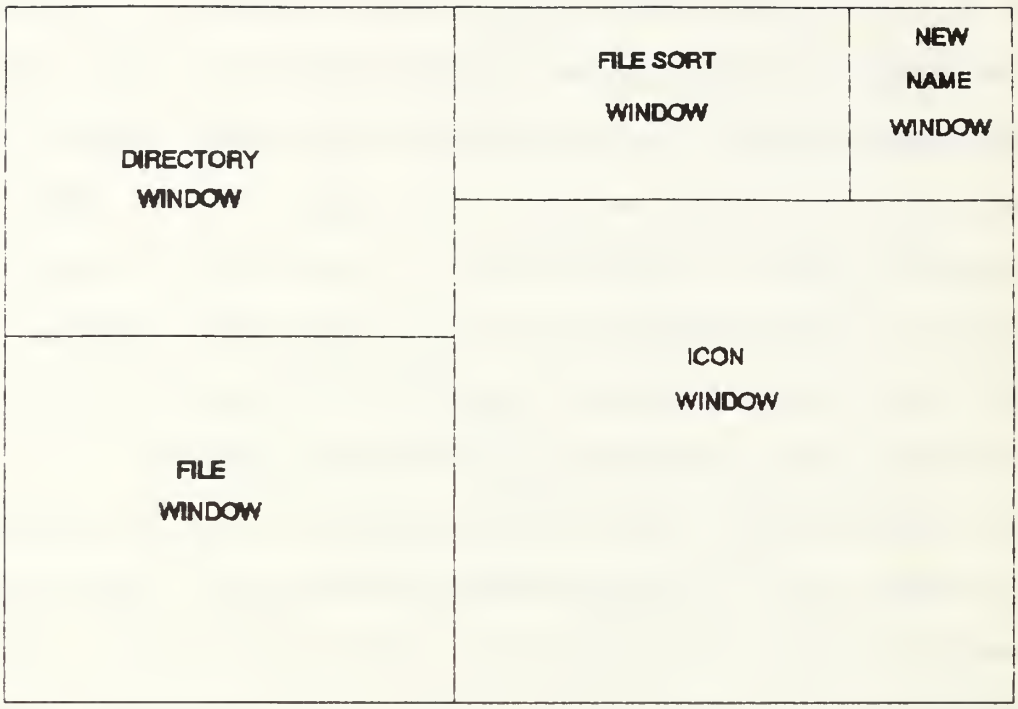


Figure 3. The Direct Manipulation Interface Top Pane and Component Windows

structure. For example, each subdirectory will be indented one space from the parent directory.

2. *File Window*

The name of the selected directory appears in reverse video (black background, white lettering) in the Directory window and the names of all files contained in the directory are displayed in the File window. Files listed in the File

window are in alphabetical order. All operations on files will be carried out using the file window.

3. *File Sort Window*

The File Sort window contains the names of all the files listed in the file window, plus additional information about these files. The File Sort window has two parts, a label and a text portion. The label, i.e., Files Sorted by Name, contains a menu which allows the files to be sorted by name, date, or size. Items can be selected from this menu by clicking the right mouse button when the arrow is over the label and selecting with the left mouse button, the desired sorting method. The File Sort window will then contain the files of the selected directory sorted by the specified method (see Figure 4).

4. *New Name Window*

The New Name window contains a list of all the names needed for new files, renamed files, and directories. The new name must be selected before an icon is selected to complete an operation on a file, if that operation involves naming the file. The system removes the selected new name after it has been used.

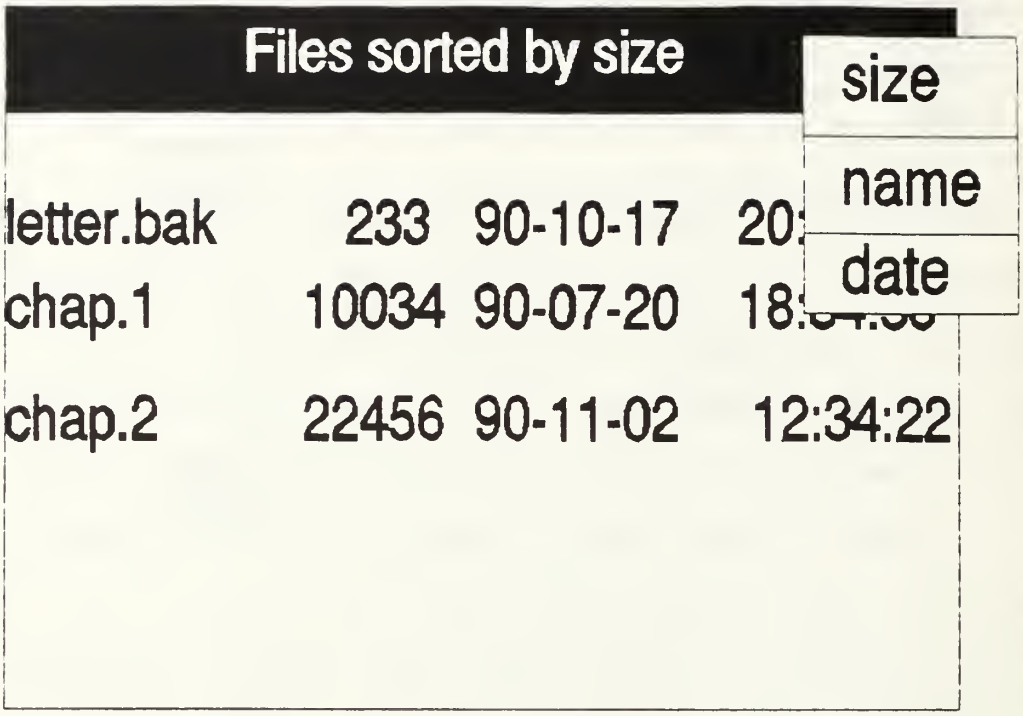


Figure 4. Example of File Sort Window, with File Names Sorted by Size.

5. *Icon Window*

The icon window is divided into three sections: File Icons, Directory Icons, and Other Icons. Using this window, files and directories can be created, copied, renamed, and deleted, as discussed below.

a. *Create File Icon*

The Create File icon creates a new file in the selected directory using the name selected in the New Name window.

b. Copy File Icon

The **Copy File** icon copies a selected file to the selected directory using the selected name in the New Name window.

c. Rename File Icon

The **Rename File** icon renames a selected file to the selected name in the New Name window.

d. Delete File Icon

The **Delete File** icon deletes the selected file from the selected directory.

e. Create Directory Icon

The **Create Directory** icon creates a new subdirectory under the selected directory using the selected name in the New Name window.

f. Directory Tree

The **Directory Tree** icon displays a graphical depiction of the directories on the drive.

g. Delete Directory

The **Delete Directory** icon deletes the selected directory from the disk. The selected directory **can not** contain **files** nor **subdirectories.**

h. Help

The **Help** icon displays the help window described below.

6. Prompter Window

When you are performing a file or directory operation a "prompter window" will appear. This window allows you to confirm or cancel the operation. When this window appears, click the **right** mouse button while the arrow is in the white portion of the window (suggested file or directory name) to call up the menu. Then select the "accept" or "cancel" option with the **left** mouse button. A prompter window also will appear when you attempt to conduct an operation without having specified all the necessary information. The needed information will appear in the white portion of the window. Remove the prompter window by selecting "cancel" from the prompter menu.

7. Help Window

The Help window provides information on window operations, icons, and window locations. The Help window can be displayed by selecting the Help icon. **Hold down** the left mouse button until a prompt appears for the left corner of the help window. **Move** the pop-up window corner to the upper left corner of the screen and release the mouse button. The lower

right corner of the window will appear and can be repositioned with the mouse. Click the left mouse button when the size of the window is at least five inches square.

The help commands appear in alphabetical order. When a command is selected, a description of the command will be provided in the right pane (text pane) of the window. The command list and text pane can be scrolled in the manner discussed earlier, press the right mouse button and drag the cursor out of the pane in the direction of the unseen text.

8. Error Windows

Error windows can appear either as a **Prompter** window or as a **Pop-up** window. A Prompter window will be a small window with a short message. Selecting an icon without all the necessary information specified will cause it to appear. It can be removed by (1) selecting the **right** mouse button when the cursor is located in the white portion of the prompter (area of short message) to obtain the menu options and then (2) selecting the "cancel" option.

If you attempt an operation that the operating system does not allow, an Error **Pop-up** window will appear in the middle of the screen. The Label, located at the top, will

contain the error message. The remaining text portion of the window may be confusing and not necessary for you to understand. To remove the window, use the mouse's left button and cursor to select a point outside the window, or select the menu with the right mouse button in the Label and then the "close" option with the left mouse button.

9. *Directory Tree*

A Directory Tree is helpful for seeing the "whole picture" of directories and subdirectories. Similar to the Directory window, a subdirectory will branch off from its parent directory. The Directory Tree window can be displayed by selecting the Tree icon. Hold down the left mouse button until a prompt appears for the left corner of the Tree window. Move the pop-up window corner to the upper left corner of the screen and release the mouse button. The lower right corner of the window will appear and can be repositioned with the mouse. Click the left mouse button when the size of the window is approximately five to six inches square.

II. YOUR TASK

You are participating in an experiment that is being conducted to evaluate the interface of a recently developed computer file management system. The purpose of the experiment is to determine the strengths and weaknesses of the interface. You must perform a series of tasks using the interface. As you perform these tasks, you will be required to "think aloud"--that is, you will say aloud what you are thinking about, any questions that you may have concerning the task, or anything related to the system that may cause you confusion. Throughout this period, your thoughts will be recorded for later analysis. It is important to remember that it is not your thought process that is being evaluated, but rather the type of concerns you encounter while attempting to learn and perform tasks using the file management system provided. Although you have been asked to verbalize your questions as they arise (and these question are vital to the final evaluation of the system), it is important that you realize that your questions probably will not be answered. Finally, if you become absorbed in the performance of the task and therefore stop verbalizing your thoughts, the observer will provide prompting.

***** STOP *****

THE OBSERVER WILL ASSIST YOU IN PROCEEDING WITH THE EXPERIMENT

V. EXPERIMENT

Complete the following operations using the procedures you have read about. Use the help screen as needed. Work at a normal pace and as accurately as possible.

1. Find the file called *plane* and copy it to a file called *aircraft* within the same directory.

2. Create a file called *car* in the *ground* directory and sort *ground* files by file size.

3. Delete the *planes* directory.

4. Find the largest file of all the directories and rename the file to large.fil.

5. The system supervisor informs us that the *ground* directory is a misnomer and should really be called the *fleet* directory. Also, having a *ground* directory in the system causes confusion among the staff. Rectify the situation.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Kishore Sengupta, Code AS/SE Naval Postgraduate School Monterey, California 93943	2
4.	Tung Bui, Code AS/BD Naval Postgraduate School Monterey, California 93943	2
5.	Barbara Treharne HQ's, U.S. Military Academy West Point, New York 10996	2

Thesis
T7925 Treharne
c.1 The impact of verbal report protocol analysis on a model of human-computer interface cognitive processing.

Thesis
T7925 Treharne
c.1 The impact of verbal report protocol analysis on a model of human-computer interface cognitive processing.

3 2768 00014751 6